João Filipe Rodrigues

Portal Grid para a Universidade do Porto

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Setembro de 2012

João Filipe Rodrigues

Portal Grid para a Universidade do Porto

Dissertação submetida à Faculdade de Ciências da Universidade do Porto como parte dos requisitos para a obtenção do grau de Mestre em Engenharia de Redes em Sistemas Informáticos

Orientadora: Prof.ª Dra.ª Inés de Castro Dutra

Departamento de Ciência de Computadores
Faculdade de Ciências da Universidade do Porto
Setembro de 2012

Resumo

Como muitas outras instituições que detêm infraestruturas *grid*, a Universidade do Porto tem recursos federados que fazem parte da infraestrutura de *grid* Europeia. Muitos investigadores da UP já usaram esses recursos com sucesso, através de várias organizações virtuais, suportadas pela a infraestrutura *grid* interna. Em particular, a Universidade do Porto suporta uma VO local (vo.up.pt) que agrega a sua comunidade académica. Nesta VO, os investigadores têm acesso a 100 cores.

Apesar do sucesso de utilização desta infraestrutura por alguns investigadores, ainda existe espaço para melhoramentos. Primeiro, existem ainda algumas comunidades internas que poderiam beneficiar destes recursos, mas não sabem como. Segundo, há várias razões que fazem com que as pessoas não utilizem a infraestrutura de *grid*: (1) a burocracia envolvida no uso de *grids* (por exemplo, obtenção de certificados ou a necessidade de ter uma conta de utilizador numa *User Interface*), (2) o *overhead* dos serviços *grid*, devido às várias camadas de *software* e aos servidores centralizados, (3) a elevada taxa de falhas das tarefas, e (4) a mudança na metodologia e ambiente para executar aplicações.

A fim de incentivar o uso desta infraestrutura e tirar proveito de todo o seu potencial, muitas soluções têm sido implementadas e adotadas. Uma delas é oferecer interfaces na forma de portais ou *gateways*, que reduza a curva de aprendizagem do utilizador, pelo menos minimizando os problemas (1) e (4) acima referenciados. Seguindo o mesmo caminho que outras instituições (p.e., Universiy of California at Los Angeles Grid Portal ou GISELA eScience Gateway e as versões anteriores do portal Genius), desenhámos e desenvolvemos um portal, o portal eScience GridUP, que oferece aos utilizadores uma interface amigável para interagir com a infraestrutura de *grid* que temos. Este portal foi construido sob experiências anteriores com novos utilizadores e no desenvolvimento e suporte de aplicações *grid* (p.e., o guia do utilizador EELA Grid, a biblioteca UbiDis, a execução de programas de lógica indutiva paralelos e outras aplicações). A principal caraterística deste portal é a facilidade com que os utilizadores podem submeter e gerir as suas tarefas. A submissão é feita através do preenchimento de um formulário e do carregamento dos ficheiros necessários (p.e., o executável, ficheiros de entrada). Além disso, os utilizadores podem gerir as suas tarefas, individualmente, numa fila de tarefas. Essa fila, permite os utilizadores gerir as suas tarefas e mostrar informação essencial, tal como *id* da tarefa, hora de submissão e de conclusão da tarefa. Só tarefas básicas (*Normal*) é que são suportadas até ao momento. O portal foi construido sob a plataforma Liferay, um portal livre e *open source* escrito em Java e distribuído sob a licença GNU Lesser General Public License e licenças proprietárias. O portal Grid UP usa a biblioteca

jLite como uma interface de ligação com a infraestrutura *grid*. Uma conta é criada para cada utilizador no portal (lado servidor), e o certificado é guardado para ser usado pelo comando *voms-proxy-init*.

Abstract

As many other institutions involved in grid infrastructures, the University of Porto (UP) has federated resources that are part of the European Grid Infrastructure. Many researchers at the UP already successfully use those resources, through various virtual organizations (VO), supported by our internal grid infrastructure. In particular, we support a local VO (vo.up.pt) that aggregates the academic UP community. In this VO, researchers have access to around 100 cores.

Despite the success of utilization of this infrastructure by some researchers, there is still scope for improvements. First, there are still several internal communities that could benefit from using those resources, but do not know how. Second, there are various reasons that prevent more people from using grid infrastructures: (1) the bureaucracy involved in using grids (for example, obtaining certificates or the need to have an account on a user interface), (2) the overheads of grid services due to the various layers of software and centralized servers, (3) the high rate of job failures, and (4) the change in methodology and environment to execute applications.

In order to leverage these infrastructures and take advantage of their full potential, many solutions have been implemented and adopted. One of them is to offer interfaces in the form of portals or gateways, which reduce the user's learning time, at least minimizing problems (1) and (4) aforementioned. Following the same track of other institutions (e.g., University of California at Los Angeles Grid Portal or the GISELA eScience Gateway and its previous version Genius), we designed and developed a user portal, the GridUP eScience Portal, to offer to users a friendly interface to interact with the grid infrastructure we have. This portal builds upon previous experiences with new grid users and on the development and support of grid applications (e.g., the EELA Grid User Guide the UbiDis library and the execution of parallel inductive logic programming and other applications). The main feature of this portal is the easy way of job submission and job management. The users can submit their jobs by filling a form and loading the files needed (e.g. executable, input files). Besides, the users can manage their jobs, individually, on a job queue. That queue allows the users to manage their jobs and show essential information, such as, job id, job submission and conclusion time. Only basic jobs are supported at the moment. The portal was built on top of LifeRay, a free and open source enterprise portal written in Java and distributed under the GNU Lesser General Public License and proprietary licenses, and uses the jLite library as an interface to call the gLite backend commands. An account is created for each user in the portal server, and the certificate is stored there to be used by the voms-proxy-init command.

À minha mãe, à minha irmã, à minha avó, restante família e amigos

Agradecimentos

Gostaria de apresentar os meus agradecimentos, acima de tudo, à Prof. Doutora Inês Dutra, ao Rui Ramos e Sergio Afonso e à Doutora Lígia M. Ribeiro por todo o apoio, disponibilidade e ótimas sugestões ao longo deste trabalho. Sem as suas orientações, o sucesso desta investigação não teria sido possível. Queria agradecer também ao projeto GISELA (Grid Initiatives for e-Science virtual communities in Europe and Latin America) pelo financiamento de uma bolsa de investigação e pela oportunidade dada.

Pretendo igualmente agradecer aos restantes professores e colaboradores da Faculdade de Ciências da Universidade do Porto por me terem proporcionado um ambiente de aprendizagem único ao longo do meu percurso académico.

Uma nota de agradecimento a Hélder Albuquerque, Cátia Almeida, Sérgio Gomes, Fábio Domingues, Leandro Fernandes, Diogo Marcelo Nogueira, Francisco Santos, Tiago Queirós, Carlos Moura, Gonçalo Martins, Daniel Mota, Marcelo Soares, Daniel Moreira, Luís Sardinha, João Teixeira, David Nabais, Filipe Brandão e António Mira pela paciência demonstrada conselhos sábios e pela amizade.

Finalmente, agradeço à minha mãe por me ter proporcionado todas as condições para a minha formação académica, à minha irmã e avó por todo o apoio prestado.

O projeto em questão foi suportado pelo projeto GISELA e pela Reitoria da Universidade do Porto. Foi ainda financiado pelo projeto GISELA, através de uma Bolsa de Iniciação Científica (BIC).

Este documento foi preparado com o processador de texto LaTeX. O sistema de citações de referências bibliográficas utiliza o estilo Vancouver.

Alguns termos presentes nesta dissertação não foram sujeitos a tradução da língua inglesa para a portuguesa pelo simples facto de estarem amplamente aceites, difundidos e até mesmo enraizados na comunidade académica que estuda computação *grid*.

Todos os endereços de Internet referenciados na bibliografia foram acedidos pela última vez durante o mês de Setembro de 2012.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Contextualização

Os conceitos e as tecnologias *grid* são relativamente recentes, cerca de 15 anos. Em 1998, Foster e Kesselman lançaram os primeiros conceitos. A necessidade de criação de *grid* advém dos esforços em juntar vários recursos de forma distribuída, isto para alcançar maior poder de computação [84]. Tal conceito era conhecido por *metacomputing*, que mais tarde foi "rebatizado" como *grid computing* por Foster. *Metacomputing* inclui a organização de grandes redes de computadores, escolhendo um critério de desenho (como peer-to-peer, ou cliente-servidor) e software de *middleware*. Sendo assim *metacomputing* é definido como uma plataforma de rede, organizada em camadas, orientada a cálculos científicos, e era bastante utilizado nas áreas de física computacional ou bio-informática [93].

O mundo moderno gera quantidades gigantescas de informação que necessita de ser processada, isto para que esta esteja disponível em qualquer lugar a qualquer hora. Para isso existem supercomputadores que podem processar tal informação e disponibiliza-la. Mas hoje em dia essa é tão grande que nem essa solução de computação (supercomputadores) consegue responder às necessidades exigidas, um exemplo disso é o LHC (Large Hadron Colider) do CERN que gera anualmente peta-bytes de informação e foi também um dos impulsionadores para a criação de infraestruturas *grid*. Para isso as organizações necessitavam de gastar centenas de milhões de euros para ter tal infraestrutura. Então a solução encontrada foi computação *grid*, composta por um conjunto de supercomputadores todos interligados em rede e geridos por um middleware (por exemplo gLite, Globus), responsável por gerir dados, tarefas etc. Assim com esta solução é possível ter uma capacidade de armazenamento e de processamento enormíssima a custos mais reduzidos, visto que esta é criada através da colaboração de várias instituições que disponibilizam os seus recursos computacionais para a infraestrutura grid. A expansão deste tipo de infraestruturas é praticamente ilimitado, o que faz com que esta seja a solução de computação mais poderosa do mundo em termos de capacidade de computação e de armazenamento [81]. De notar que para aplicações que exijam alto desempenho o melhor poderá ser um

supercomputador, devido à sua natureza homogénea, desde que este tenha capacidade de processamento suficiente para tal.

Para ilustrar melhor este ambiente, existem vários cenários em que computação *grid* é fundamental para a sua concretização, por exemplo:

- Grupo de cientistas a estudar a camada de ozono da atmosfera. Esses estudos necessitam de colecionar grandes quantidades de dados experimentais e ter uma infraestrutura distribuída que tenha grande capacidade de armazenamento. Normalmente essas dados estão dispersos geograficamente, então irão ser guardados num dispositivo de armazenamento mais próximo, para posteriormente serem acedidos. O acesso a esses dados é feito de forma eficiente, visto que esses são pré-processados fazendo com que a transferência seja de menores dimensões [81].

- Estudo sobre desastres naturais como, por exemplo, derrames químicos. O objetivo é simular a melhor maneira do controlo do desastre recorrendo à colaboração de vários departamentos em termos computacionais. A necessidade de grande poder computacional advém da simulação de muitos modelos computacionais relacionados com a propagação do derrame, efeito do tempo no derrame e determinar o impacto na saúde humana e dos animas selvagens. Graças a estes modelos, as autoridades poderão atuar da melhor forma de modo a minimizar as perdas da vida e também proteger o ecossistema da melhor maneira [84].

Sendo assim, computação *grid* oferece todo um conjunto de possibilidades que permite resolver os problemas enunciados acima. Sem esta infraestrutura seria praticamente impossível resolver os problemas acima, visto que é necessário grande poder de computação e também é necessário que toda a infraestrutura esteja espalhada por todo o mundo para que possa recolher os dados de forma mais eficiente [81]. Podemos imaginar, no melhor cenário, computação grid como vários supercomputadores espalhados por todos os países do mundo, em que cada um pode fornecer poder computacional que não necessita no determinado momento. Porém o mais importante é a capacidade de partilha de dados, por exemplo, recolha de dados a partir de sensores que estão distribuídos por todo o mundo. Cada sensor poderá comunicar com a máquina grid mais próxima e esta posteriormente poderá fazer algum processamento dos dados para que depois se possa juntar todos os resultados obtidos de forma mais eficiente. O processamento dos dados serve para que o tamanho dos mesmos seja drasticamente menor. Como podemos ver esta colaboração é importante para que possamos atingir um objetivo comum de forma menos dispendiosa, visto que há um auxilio por parte das várias organizações para que isso possa ser concretizado.

Do ponto de vista do utilizador, a infraestrutura possui algumas vantagens e desvantagens que serão descritas de seguida. As vantagens são as seguintes:

- Poder computacional: A infraestrutura oferece ao utilizador grande poder computacional para resolver os mais variados problemas das diferentes áreas de investigação.

- Sem custo: Os utilizadores para terem acesso à infraestrutura *grid* só necessitam pertencer a uma instituição que colabore com esta. Posteriormente necessitam também de um certificado pessoal e da autorização para tal.

- Capacidade de armazenamento: Toda a infraestrutura apresenta grande capacidade de armazenamento. Cada utilizador poderá guardar o necessário, criar réplicas de dados e mover também todos esses dados sempre que necessário. Este inclusive poderá mover dados necessários a uma tarefa, o mais próximo possível onde esta se encontra em execução. Para grandes quantidades de dados isto permite minimizar a comunicação permitindo assim diminuir o *overhead* da aplicação.

- Colaboração: Os utilizadores poderão ter acesso a dados pertencentes a outros utilizadores. Esta partilha é bastante importante para que projetos de investigação possam progredir rapidamente reaproveitando trabalho já efetuado por outros.

- Recuperação de falhas: Tarefas submetidas, por vezes, podem falhar por alguma razão. Para colmatar isso e para que o utilizador não necessite de interagir com a infraestrutura ou fazer resubmissão manual, a infraestrutura *grid* recupera da falha ocorrida, reiniciando automaticamente todo o processo de submissão e execução da tarefa.

As vantagens enunciadas, do ponto de visto do utilizador, são atrativas. Para o utilizador comum não interessa onde a tarefa vai executar, este só quer receber os resultados de forma fácil. Para isso, existe toda uma infraestrutura que permite que tal aconteça. Porém para os utilizadores menos experientes, o uso desta infraestrutura pode tornar-se complexo ou tedioso, visto que são necessários alguns procedimentos com os quais os utilizadores menos experientes poderão não estar muito familiarizados. Portanto, algumas das desvantagens deste ambiente são:

- Linha de comandos: Para a utilização de linha de comandos é necessário algum conhecimento sobre como usar sistemas operativos da família UNIX. Para utilizadores que não estejam familiarizados com estes, poderá ser complicado a sua utilização, visto que a sua adaptação ao sistema poderá ser um pouco custosa. Para utilizarem esta ferramenta é necessário a leitura de um tutorial sobre linha de comandos UNIX e também a leitura dos manuais dos comandos a utilizar. Para muitos utilizadores isto é uma grande barreira, porque estes não estão dispostos a despender tanto tempo na aprendizagem do sistema, exceto o caso de estes terem mesmo a necessidade de utilizar a infraestrutura *grid*.

- Gestão do certificado pessoal: Este processo é burocrático, o que leva os utilizadores a pensarem duas vezes em utilizar *grid*. Para o pedido do certificado, o utilizador tem a necessidade de se deslocar a uma autoridade certificadora ou autoridade de registo, preencher um formulário e esperar depois cerca de 24 horas pela resposta. Além desta burocracia, existe outra barreira que é a submissão do certificado pessoal via linha de comandos para uma *User Interface*, o que poderá ser difícil para quem não tem conhecimento de sistemas operativos Linux.

- Memorização de comandos: Os utilizadores, numa fase inicial, poderão necessitar verificar os manuais dos comandos diversas vezes, visto que estes são um pouco extensos e também das diversas combinações que poderão existir. Isto faz com que o processo de submissão e também de adaptação à infraestrutura seja um pouco morosa.

Como podemos ver acima, existem algumas limitações que fazem com que os utilizadores não utilizem as infraestruturas de *grid*. Essas limitações são devido principalmente a utilização de uma interface de linha de comandos e da alta taxa de falhas de tarefas existentes na infraestrutura. Para isso é que muitas instituições criaram portais para que estes a possam utilizar de uma forma mais fácil e menos burocrática.

Existem algumas instituições que criaram o seu próprio portal *grid* para incentivar o uso das infraestruturas e também para facilitar a vida a todos os utilizadores que a desejam utilizar. Os portais mais importantes que atualmente existem são: o portal UCLA da Universidade de Califórnia [53], o portal GENIUS do projeto EGEE "Enabling Grids for E-sciencE" financiado pela comissão europeia [77], o portal P-GRADE desenvolvido pelo laboratório MTA-SZTAKI, Hungria [73] e o portal GSG "GISELA Science Gateway" desenvolvido pelo instituto de física nuclear em Catania, Itália [66]. Os portais vieram revolucionar um pouco a forma de utilização de toda a infraestrutura *grid*, visto que o objetivo, dos portais, é ocultar detalhes técnicos inerentes. Sendo assim estes permitem criar um meio de comunicação com a infraestrutura de forma transparente e fácil. Os portais *grid* tentam reproduzir todas as funcionalidades que os utilizadores teriam numa *User Interface*, mas de forma gráfica e amigável.

1.2 Motivação

Portanto, a motivação inerente à criação do portal Grid UP prende-se com a tentativa de aumentar a visibilidade da infraestrutura *grid* da Universidade do Porto, assim como oferecer uma interface de fácil utilização a toda a comunidade existente. Tal como outras instituições, que estão envolvidas em projetos *grid*, a Universidade do Porto possui alguns recursos computacionais que poderão ser utilizados pelo bem de muitos investigadores existentes. Porém já existem instituições que possuem portais *grid* e estes encontram-se disponíveis a instalar e utilizar. A opção de não utilizar os portais já existentes recai sobre a incapacidade destes de realizar os objetivos propostos que serão enumerados na próxima secção. Alguns destes portais apresentam problemas, tais como, utilizam tecnologias antigas e obsoletas, o que dificulta a sua manutenção e expansão, pelo menos um deles foi descontinuado e outros limitam-se a fornecer aplicações pré-construídas.

Através do portal Grid UP, o utilizador tem um acesso completamente transparente à infraestrutura permitindo assim que estes executem ações de forma gráfica, que por norma são mais intuitivas do que a utilização de linhas de comandos. A linha de comandos, no entanto tem flexibilidade e um grande leque de opções podendo ser útil para utilizadores mais avançados. À medida que o tempo avança e com o crescente ritmo de investigação

essas necessidades tendem a ser suprimidas e num futuro próximo teremos com certeza interfaces gráficas com maiores número de opções, mais eficientes e dinâmicas.

Como mencionado anteriormente o incentivo para utilizar a infraestrutura *grid* da Universidade do Porto passa por oferecer uma interface de fácil utilização e intuitiva. Muitos dos utilizadores que utilizam estas infraestruturas não são das áreas de Ciência de Computadores ou Informática o que dificulta o seu uso e também é uma das razões para que esta seja pouco utilizada. Então a solução passa por esconder toda a complexidade existente e oferecer algo que reproduza as ações necessárias de forma mais intuitiva e fácil, isto para que a utilização de grid possa ser acessível a todos os utilizadores que desejarem a utilizar.

1.3 Objetivos

Nesta secção apresentamos um conjunto de objetivos, os quais o portal *grid* da Universidade do Porto se deve reger:

- Atender aos requisitos dos utilizadores e das tarefas: Todos os utilizadores deverão ser capazes de submter e gerir todas as suas tarefas. O portal deverá também suprimir as necessidades dos utilizadores a nível do tipo de tarefas a submeter, ou seja, os utilizadores têm a necessidade de um portal que lhes permite executar tarefas que exijam grande poder de computação e de armazenamento.
- Melhorar a visibilidade do serviço *grid*: O número de utilizadores atualmente que utilizam a infraestrutura *grid* é reduzido. Com o portal, o objetivo é abranger um maior leque de utilizadores a utilizarem tal infraestrutura. Sendo assim, o portal deverá ser uma forma de incentivo para que todos os tipos de utilizadores (tanto os com conhecimento tanto os sem conhecimento em infraestruturas deste tipo) se sintam atraídos pela sua utilização.
- Autenticação: Antes de mais, os utilizadores deverão realizar autenticação por forma a terem acesso a todas as funcionalidades do portal. Essa autenticação deverá ser federada com as credenciais da plataforma *sigarra*, isto para que só utilizadores da Universidade do Porto possam a utilizar e também para que estes não necessitem de criar novas credenciais visto que normalmente os utilizadores têm dezenas delas.
- Pedido e submissão de certificados: Este serviço serve para reduzir a burocracia existente no pedido de certificados. Todos os utilizadores devidamente autenticados deverão ser capazes de pedir e instalar o seu certificado pessoal via portal. O pedido deverá ser um processo automático sem a necessidade de deslocação por parte do utilizador.
- Submissão de tarefas: Qualquer utilizador deverá conseguir submeter tarefas diretamente via portal sem a necessidade de conhecimentos técnicos de grid. Deverá ser apresentada uma interface simples e "limpa" para que esta seja fácil de usar, intuitiva e também para que todos os utilizadores se sintam atraídos. Para utilizadores mais avançados estes deverão ser capazes de aceder e criar outras funcionalidades.

- Ajuda: O portal deverá fornecer toda a ajuda possível para que os utilizadores possam tirar as suas dúvidas.

- Fila de tarefas: Neste serviço o utilizador deverá ter a capacidade de visualizar todas as tarefas submetidas e fazer ações sobre elas individualmente. Além disso, deve ser capaz de inspecionar resultados das suas tarefas.

- Informação: O portal deverá mostrar todas as componentes que compõem a infraestrutura *grid*, a sua respetiva localização e as organizações virtuais existentes. Deverá também mostrar alguma informação elucidativa, explicando o que é *grid*, para que serve um portal, vídeos de utilização, notícias sobre a Universidade do Porto. Por fim o portal deverá também mostrar informação sobre o estado atual da infraestrutura *grid*, como por exemplo o número de tarefas em execução, número de tarefas em fila de espera, balanceamento de carga dos *clusters*, informação sobre armazenamento entre outros.

Os objetivos acima referenciados são essenciais para que todos os tipos de utilizadores tenham as condições mínimas e de "conforto" para executar tarefas em infraestruturas *grid* de forma simples. Os objetivos propostos oferecem toda uma interface ao utilizador para que este submeta todas as suas aplicações e visualize todo o seu progresso de forma rápida e fácil. Este também terá a possibilidade de diminuir a burocracia inerente aos certificados pessoais automatizando todo o processo "sem sair do lugar".

1.4 Contribuições

Neste trabalho, a principal contribuição é o próprio portal Grid UP. Consideramos também como contribuição (1) a revisão e discussão das tecnologias utilizadas para a construção de portais e (2) a revisão dos portais de grid disponíveis na literatura. Além disto, este trabalho contribuiu com uma publicação na conferência GISELA-CHAIN, com o seguinte título: *The GridUP Portal*.

O público alvo do portal Grid UP, são todos aqueles que não possuem conhecimentos técnicos de como utilizar a infraestrutura e de como gerir todas as suas tarefas manualmente em ambiente *shell*. Sendo assim o portal não deverá apresentar grandes detalhes técnicos que dificultem a sua usabilidade. Então este deverá servir como forma de publicidade para incentivar todos os investigadores a utilizarem recursos que estão disponíveis dentro da sua própria instituição. O portal está recetivo também a todos os utilizadores mais avançados, pois este apresenta todo um conjunto de atributos que estes podem definir posteriormente (desde que o tipo de tarefas seja "Normal"). Como é de esperar, para estes utilizadores, o portal deverá ao longo do tempo acrescentar novas funcionalidades e abranger mais conteúdo para que este tipo de utilizadores possam também usufruir de um ambiente amigável e simples, sem a necessidade de um ambiente *shell*. Isto porque os serviços existentes são atualmente insuficientes para estes tipos de utilizadores preferindo assim usar um ambiente *shell* que é uma ferramenta bastante poderosa e dinâmica.

1.5 Organização da Dissertação

Esta dissertação está organizada da seguinte forma. No Capítulo 2 falamos sobre infraestruturas de *grid*. Este descreve o conceito de *grid* e também a sua arquitetura definida pelo OGF (Open Grid Forum). Além disso, este capítulo fala sobre a infraestrutura EGI, na qual a Universidade do Porto está inserida. Este também descreve as componentes mais importantes do *middleware* gLite separadamente.

O Capítulo 3 narra essencialmente a evolução dos portais *grid*, como, por exemplo, a evolução das tecnologias utilizadas. Este fala também sobre os portais existentes mais importantes, onde são discutidos basicamente a sua arquitetura e algumas vantagens e desvantagens.

O Capítulo 4 fala sobre os requisitos que levaram à construção do portal *grid* da Universidade do Porto e também sobre as tecnologias utilizadas na sua construção. Além disso, este capítulo, descreve todas as funcionalidades suportadas atualmente pelo portal.

No Capítulo 5 descrevemos a arquitetura do portal Grid UP. Este fala essencialmente sobre a forma de organização e as interações existentes com a base de dados, o sistema de ficheiros, utilizador entre outras.

No Capítulo 6 é feita uma reflexão sobre todo o trabalho realizado e também são apresentados alguns pontos importantes sobre o trabalho futuro a efetuar para melhorar a usabilidade do portal e também para acrescentar outros serviços essenciais.

Por fim no Anexo A descrevemos os casos de uso para cada serviço específico existente. Para cada caso existem imagens que ilustram a interface e também textos explicativos que as complementam.

Capítulo 2

Infraestruturas de *Grid*

2.1 Introdução

Neste capítulo, descrevemos o conceito de *grid* e de organização virtual. Além disso mostraremos também a arquitetura *grid* base definida pelo OGF, alguns serviços da infraestrutura EGI e também uma descrição do *middlaware* gLite. Este capítulo está organizado da seguinte forma. A secção 2.2 define o conceito de computação *grid* e também o seu objetivo. A secção 2.3 descreve o que é uma organização virtual e referencia alguns exemplos existentes. A secção 2.4 descreve as diferentes camadas da arquitetura *grid*. A secção 2.5 ilustra alguns serviços, recursos existentes na infraestrutura EGI e também os seus participantes. A secção 2.6 descreve separadamente as principais componentes do *middleware* gLite e também alguns serviços, além de fornecer uma breve descrição sobre a linguagem utilizada para criação de tarefas no ambiente *grid* gLite. Para finalizar a secção 2.7 descreve os diferentes ciclos de vida a que uma tarefa submetida está sujeita.

2.2 O que é Computação *Grid*

O termo computação *grid* foi inventado por Ian Foster [84] e pode ser definido como um conjunto de recursos computacionais pertencentes a diferentes domínios administrativos, que tentam alcançar um objetivo comum [74]. Podemos pensar em computação *grid* como um sistema distribuído com tarefas não interativas, envolvendo grandes quantidades de ficheiros dos mais diversos tamanhos. De notar que, também existem propostas para *grids* interativas como por exemplo IntEuGrid "Interactive European Grid" em que o seu principal objetivo é fornecer ao investigador poder computacional elevado de forma transparente utilizando recursos distribuídos [1]. Uma tarefa interativa é a aquela em que o utilizador pode modificar os seus parâmetros em tempo de execução sem reiniciar a tarefa, já uma tarefa não interativa é aquela que o utilizador submete e não pode alterar nenhum parâmetroem tempo de execução. Tal como os sistemas distribuídos, computação

grid envolve também grandes quantidades de processamento o que faz com que estes sejam bastante parecidos. No entanto existem algumas diferenças que permitem distinguir computação *grid* de sistemas distribuídos [87], embora essas diferenças só sejam visíveis a nível técnico, visto que para o utilizador é praticamente a mesma coisa. De notar que os sistemas distribuídos enunciados são clusters para HPC "High Performance Computing". Segue então algumas das principais diferenças entre os dois sistemas enunciados:

- Dispersão geográfica. Normalmente um sistema *grid* está disperso por diferentes domínios administrativos (diferentes países), enquanto um sistema distribuído está concentrado num único local.
- Heterogeneidade. *Grid* é um sistema heterogéneo que contém diversos recursos computacionais diferentes uns dos outros. Já um sistema distribuído é homogéneo, pois todos os recursos são semelhantes.
- Virtualização. Semanticamente um sistema *grid* é diferente dos sistemas distribuídos convencionais, pois esses (sistemas *grid*) são construídos sob uma camada virtual. A camada virtual pode ser separada em duas funcionalidades essenciais: abstração de recursos e de utilizadores, onde todo o mundo físico é separado do mundo virtual.
  - Abstração de utilizadores: Num sistema distribuído convencional, um utilizador terá de ter uma conta local para poder utilizar os recursos distribuídos. Em *grid* isso não acontece, pois todos os utilizadores são identificados através de um certificado. Através dessa identificação, cada utilizador poderá utilizar os recursos computacionais temporariamente. Esses recursos serão atribuídos automaticamente e toda a infraestrutura irá gerir todas as tarefas submetidas em nome do utilizador em questão.
  - Abstração de recursos: Num sistema distribuído, a submissão de tarefas é feita fisicamente nos servidores, onde cada utilizador terá de ter uma conta local para poder executar seja o que for. Num sistema *grid* a submissão é feita de forma virtual em que cada utilizador define uma tarefa com determinados requisitos e a infraestrutura tentará escolher os melhores recursos para tal (escalonamento). Todo o processo de escalonamento e feito de forma virtual. Existe um sistema de informação responsável por recolher dados atualizados dos recursos computacionais, recorrendo a esses dados, o escalonador vai escolher os melhores recursos disponíveis.

Como se pode verificar, existem algumas diferenças entre sistemas *grid* e sistemas distribuídos. Ambos são orientados para executar aplicações que requeiram alto desempenho, mas a natureza virtual de um sistema *grid* permite implementar aplicações colaborativas (ambientes partilhados), enquanto num sistema distribuído convencional todo o ambiente é muito rígido é físico para à colaboração entre utilizadores. No entanto existe a possibilidade também de haver colaboração [87].

Ao longo dos anos têm sido criadas várias definições de *grid* por diferentes pessoas, mas em 2001, Foster, Kesselman e Tuckle criaram uma definição de *grid* que ainda é usada nos

dias de hoje e que define de uma forma abstrata o que é computação *grid*. Essa definição é [97]:

- "The grid is coordinated resource sharing and problem solving in dynamic, multi institutional virtual organizations".

Para perceber melhor o que é um sistema *grid*, Ian Foster produziu uma pequena lista a enunciar a forma de como os ambientes *grid* deveriam ser construídos [81].

- "We should avoid building grid systems with a centralized control. Instead, we must provide the necessary infrastructure for coordination among the resources based on respective policies and service-level agreements".
- "The use of open standards provides interoperability and integration facilities. These standards must be applied for resource discovery, resource access and resource coordination".
- "We should provide quality of service (QoS) requirements necessary for the end-user community. These QoS validations must be a basic feature in any grid system, and must be done in congruence with the available resource matrices. These QoS features can be (for example) response time measures, aggregated performance, security fulfillment, resource scalability, availability, autonomic feature such as event correlation and configuration management, and failover mechanisms".

A lista acima dada, remete-nos para a forma de como uma infraestrutura *grid* deve ser construida e define os requisitos que ela deve oferecer ao utilizador.

2.2.1 Para que Serve?

De uma forma geral, podemos dizer que computação *grid* serve para guardar grandes quantidades de dados, normalmente quantidades que os supercomputadores não suportam, e para fazer quantidades enormes de processamento. Podemos pensar em *grid* como um conjunto de computadores dispersos geograficamente, heterogéneos, que estão à nossa disposição para que possamos utilizar sempre que necessitemos de recursos de *hardware* mais exigentes.

2.3 O que é uma organização virtual?

Para podermos utilizar uma infraestrutura *grid* deveremos pertencer a uma organização virtual (VO) e ter um certificado válido (devidamente assinado por uma autoridade certificadora válida) que nos identifique na infraestrutura *grid*. A autoridade certificadora LIP (Laboratório de Instrumentação e Física Experimental de Partículas), emite certificados no formato PKCS12. Um certificado PKCS12, é um certificado X.509 baseado em PKI

(Public Key Infrastructure). Certificados X.509 especificam standards para certificados de chave pública, lista de revogação de certificados, entre outros. Uma VO é um sistema virtual que define um conjunto de regras e permissões. Sendo assim, uma VO permite a determinado utilizador, utilizar recursos computacionais que estejam acessíveis a partir dela.

Podemos definir organização virtual (VO) da seguinte maneira: "dynamic collection of multiple organizations providing coordinated resources sharing" [85]. Em outras palavras, uma VO é um conjunto de indivíduos e instituições que partilham recursos baseado em regras de partilha. Numa VO, essa partilha é altamente controlada, já que os recursos dos fornecedores e consumidores são definidos de forma muito clara e cuidadosa, visto que estes podem ser partilhados [76].

Pertencer a uma VO é uma forma de organizar grupos diferentes de trabalho em que cada grupo representa o esforço de um conjunto de indivíduos para tentar alcançar a resolução de um determinado problema. O "segredo" da VO é a colaboração, ou seja, capacidade de partilhar recursos computacionais e também todos os dados que tenham vindo a ser gerados para que possam ser reutilizados. Eis alguns exemplos de VOs [97]:

- Biologia
- Física das altas energias
- Medicina
- Química
- Ciências da terra

Como é de prever, diferentes VOs poderão variar bastante o seu propósito, tais como, o âmbito de trabalho, tamanho, duração, estrutura, comunidade e sociologia [76]. Isto permite que uma comunidade se concentre na resolução de um determinado problema e que tenha muito do material partilhado e disponível. Na Figura 2.1 é apresentado um exemplo de topologia de organizações virtuais. Nesta figura, podemos identificar três organizações virtuais distintas, onde algumas instituições (ou pessoas) participam em mais que uma VO. Cada VO define os recursos que serão compartilhados e quem pode partilhar estes recursos.

Ao longo do tempo a comunidade *grid* tem dedicado um grande esforço em melhorar as organizações virtuais, tais como, protocolos, serviços, ferramentas, etc. As VOs são muito importantes em *grid* pois eliminam a natureza estática de um domínio administrativo e dinamizam grupos de trabalho pertencentes a vários domínios, virtualizando um conjunto de regras que permite tal flexibilidade.

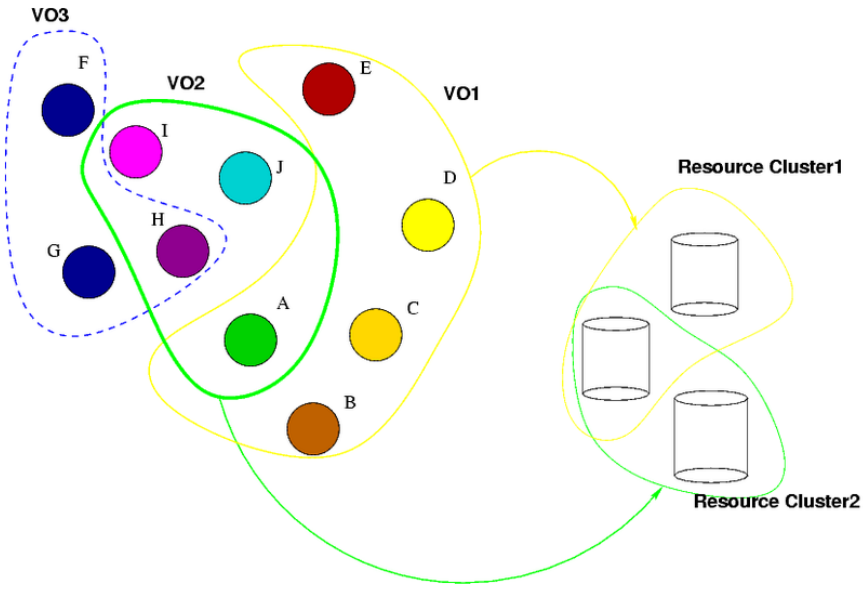

Figura 2.1: Diagrama de organizações virtuais [58].

2.4 Arquitetura grid

Nesta secção irá ser descrita a arquitetura *grid* OGSA (Open Grid Service Architecture) definida pelo OGF (Open Grid Forum). OGF é uma comunidade constituída por milhares de pessoas (investigadores e engenheiros), com o propósito de criar *standards* para computação *grid* [39]. Esta arquitetura *grid* organiza as suas componentes em camadas. Estas têm características em comum e partilham comportamentos fornecidos pela camada imediatamente abaixo [76]. A estrutura de camadas *grid* pode ser vista na figura 2.2. Segue-se então uma breve explicação sobre a função de cada camada.

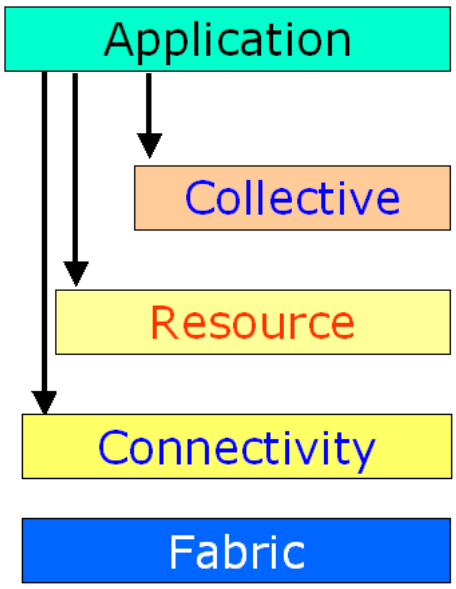

Figura 2.2: Estrutura de camadas *grid* OGSA [40].

2.4.1 Camada *Fabric*: Interfaces para controlo local

A camada *fabric* fornece o acesso partilhado a recursos locais de *hardware*, por exemplo recursos computacionais, sistemas de armazenamento, catálogos, recursos de rede e etc. Esta camada partilha os recursos locais, tanto físicos como lógicos. Isto é possível recorrendo a protocolos específicos na gestão de recursos, o que permite criar operações de partilha mais sofisticados. Como dito anteriormente, esta camada é responsável pela partilha de recursos locais, e estes podem ser caraterizados como [76]:

- Recursos computacionais: Recursos necessários para a inicialização de programas, monitorização e controlo. Esta camada permite alocar recursos previamente antes de uma tarefa ser submetida para a máquina local. Ela é também responsável por fornecer toda a informação interna a um sistema de informação. Essa informação poderá ser, por exemplo, características do *hardware* e *software*, a carga (*load*) da máquina, etc.

- Recursos de armazenamento: Gestão de ficheiros, colocar e obter ficheiros. Estes recursos controlam a leitura e a escrita de ficheiros e são também responsáveis pela alocação de espaço em disco para uma possível transferência.

- Recursos de rede: Mecanismos de controlo sobre os recursos de rede local. Este permite alocar recursos e priorizar transferências.

- Repositórios de código: É uma forma especializada de gestão de versões de código, tal como CVS "Concurrent Version System".

- Catálogos: É uma forma especializada de gestão de recursos em disco, tais como, operações de atualização, pedidos de informação, etc.

Como visto acima, a camada *fabric* é responsável pela alocação prévia de todos os recursos físicos/lógicos para uma tarefa/dados para que possam ser executados/guardados com sucesso. Esta também é importante para oferecer informações de todos os seus recursos a um sistema de informação.

2.4.2 Camada *Connectivity*: Comunicação e Segurança

A camada *connectivity* define os protocolos de comunicação e segurança usados em *grid*. Esta é responsável pelos protocolos de troca de dados existentes e define também protocolos de autenticação e segurança (criptografia) para verificação dos utilizadores e recursos. Toda a comunicação normal entre máquinas como, roteamento, DNS (Domain Name Service), é feita pela camada de rede da pilha protocolar TCP/IP. Esta camada é uma extensão da camada de rede TCP/IP. Em *grid* as extensões são definidas por questões de segurança, permitindo assim complementar a camada existente. Eis, então, algumas delas: [76]:

- *Single sign-on*: Os utilizadores são capazes de se autenticar uma única só vez e ter acesso a um conjunto de recursos da *grid* durante um certo período de tempo.
- Delegação: Os utilizadores são capazes de correr um programa com as mesmas permissões que detêm para aceder aos recursos. Por outras palavras, o utilizador delega um conjunto de direitos a um programa.
- Integração com várias soluções de segurança local: Cada recurso local tem um conjunto de políticas de segurança, então a infraestrutura *grid* deverá ser capaz de interoperar com essas. Os recursos locais não deverão deixar fazer uma substituição total das regras de segurança, mas sim permitir um mapeamento no ambiente local para uma tarefa externa.
- Relações baseadas em confiança de utilizadores: Como *grid* usa VOs, então cada utilizador pode dar permissões de partilha a outros utilizadores. Então se um utilizador quer aceder a um recurso partilhado, este deverá conseguir aceder sem qualquer imposição de segurança do domínio a que a informação pertence. Por outras palavras, os administradores de cada domínio não deverão ter a necessidade de interagir no processo.

As soluções de segurança *grid* devem fornecer um suporte de comunicação flexível para que todo o sistema internamente possa interagir sem qualquer intervenção humana, mas que assegure toda a proteção necessária como, autenticação, autorização e delegação de direitos.

2.4.3 Camada *Resource*: Partilha de Recursos Individuais

A camada *resource* é responsável por definir protocolos, APIs (Application programming interface) e SDKs (Software development kit). Os protocolos definidos por esta camada são, negociação de segurança, inicialização de tarefas, monitorização, controlo e contabilidade. Para que isto seja possível, os protocolos desta camada chamam funções definidas na camada *fabric* para aceder e controlar recursos. Os protocolos desta camada podem ser divididos em duas categorias [76]:

- Protocolos de informação: Estes são usados essencialmente para obter informação sobre a estrutura e o estado de um recurso. (por exemplo, carga corrente, configuração etc).
- Protocolos de gestão: Estes são usados, mais especificamente para negociar o acesso a recursos. Eles são usados na criação de ambientes de execução nos diversos recursos e também fornecem suporte à monitorização de estado e controle.

Como vimos, esta camada é a responsável pela definição de alguns protocolos de comunicação, mais precisamente para obter informação e para alocar recursos. A camada *resource* disponibiliza algumas APIs e SDKs, normalmente C e JAVA, para a utilização destas tecnologias o que facilita a integração de vários recursos na *grid*.

2.4.4 Camada *Collective*: Coordenar múltiplos recursos

Ao contrário da camada *resource*, que gere todos os recursos individualmente, a camada *collective* gere um conjunto de recursos como um todo. Esta camada, tal como a anterior, fornece protocolos e serviços que gerem os recursos, mas de uma forma global. Eis alguns comportamentos adicionados a esta camada [76].

- Serviços de diretório: Este serviço permite que os utilizadores de uma VO, possam perguntar por recursos pelo nome, tipo, etc.
- Co-alocação, escalonamento e serviços *brokering*: Permite que os utilizadores façam alocação de vários recursos para um dado propósito e escalonar as tarefas para os recursos alocados.
- Monitorização e serviços de diagnóstico: Suporta a monitorização dos recursos e faz alertas para casos de falhas, intrusões e sobrecarga.
- Serviços de replicação de dados: Gere os recursos de armazenamento de uma VO, maximizando a performance de acesso.
- Sistema de programação *Grid-Enable*: Ativam o suporte para modelos de programação familiares a serem usados na *grid*, como por exemplo MPI (Message Passing Interface). MPI é uma biblioteca de parelização de programas de forma distribuída (memória distribuída) [61].

A camada *collective* é responsável pela gestão de protocolos e serviços que gerem os recursos como um todo. Tal como a camada *resource*, esta também fornece SDKs e APIs que podem ser utilizadas na interligação com as aplicações. As componentes desta camada podem ser adaptadas conforme as necessidades, por isso a disponibilização de SDKs e APIs [76].

2.4.5 Aplicação

A camada aplicação contém as aplicações dos utilizadores que operam dentro de uma VO. Esta camada não é nada mais nada menos que a chamada de serviços definidos pelas camadas inferiores [76].

2.5 Infraestrutura EGI (European Grid Iniciatives)

EGI é uma infraestrutura de *grid*, financiada com dinheiros públicos europeus, com o objetivo de oferecer aos cientistas europeus um conjunto de recursos computacionais poderosos por forma a que estes possam conduzir as suas pesquisas com sucesso. Esta infraestrutura é constituída por supercomputadores, dispersos geograficamente, ligados através de redes de alta performance. A infraestrutura EGI permite aos investigadores

partilhar informação de forma segura, analisar dados de forma eficiente e colaborar com outros cientistas que utilizam a infraestrutura e que estão geograficamente distantes. Sendo assim, a EGI conduz a investigação e inovação a outro nível, visto ser um caminho mais rápido e eficiente para tal, permitindo assim beneficiar a economia e a sociedade [88]. A colaboração nos tempos de hoje é algo muito apreciado pelas empresas e pelos centros de investigação, já que com esta conseguiremos modernizar a ciência que por sua vez irá ser o motor para o crescimento económico das nações. A Figura 2.3 mostra todos os países que contribuem com recursos computacionais para a infraestrutura EGI.

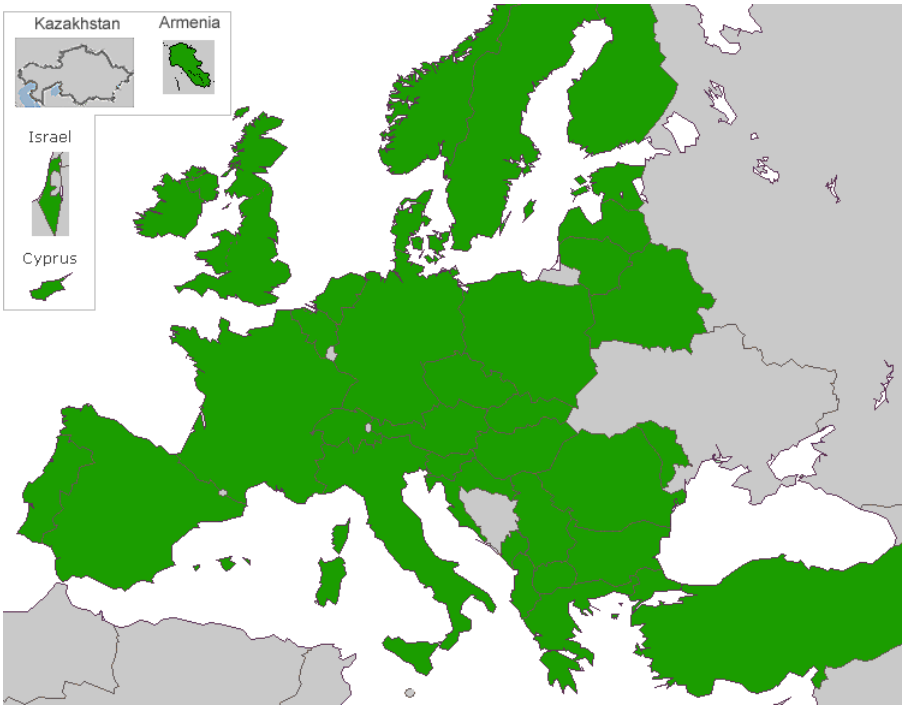

Figura 2.3: Países pertencentes à infraestrutura EGI [11].

A infraestrutura EGI é coordenada através do esforço das várias organizações pertencentes ao projeto. Estas são responsáveis pela supervisão da informação, suporte à comunidade, suporte à gestão avançada de rede, interoperabilidade entre diferentes *middlewares* e estatísticas. A figura 2.4 ilustra a forma como são organizadas as diferentes funções por cada instituição. Sendo assim, cada instituição é responsável pela configuração e gestão de determinados serviços. Esta cooperação é importante para que a infraestrutura esteja sempre funcional, visto que está dispersa por vários países da Europa.

2.5.1 Infraestrutura de serviços

Existe um conjunto de indivíduos responsáveis pela manutenção e coordenação de alguns serviços, para que toda a infraestrutura esteja funcional 24 horas por dia 7 dias por semana. Eis alguns serviços geridos diariamente pela comunidade:

- Configuração das bases de dados: Local onde é guardada toda a informação estática,

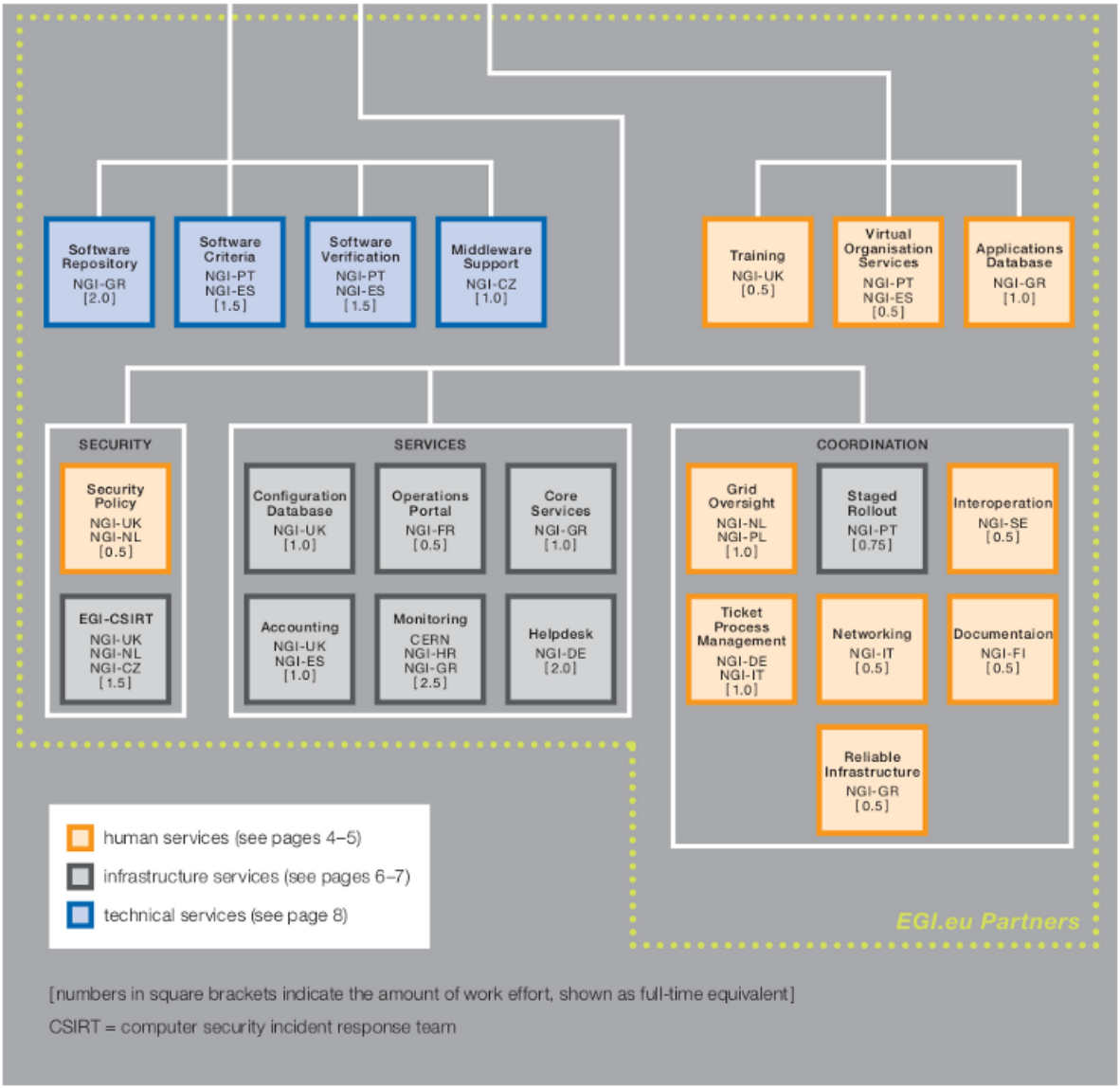

Figura 2.4: Organização das actividades [88].

como por exemplo os recursos *grid*. Esta também serve como fonte de informação para um conjunto de serviços, por exemplo *mailing lists*.

- *HelpDesk*: Responsável por ajudar em qualquer dúvida dos utilizadores. Existe uma equipa de profissionais vocacionados para dar a solução para qualquer problema que possa surgir.
- Monitorização da infraestrutura: Serviço responsável pela monitorização de todo o trabalho da *grid*. Este periodicamente faz teste de nível de funcionalidade à infraestrutura e gera relatórios sobre a atividade.
- Portal de operações: Portal responsável por monitorizar a infraestrutura de forma gráfica. O tipo de informação que este monitoriza é, por exemplo verificar as VOs que estão ativas ou não e em que local, falhas existentes, etc.
- *Staged rollout*: Serviço responsável por todas as atualizações, testes e verificação de *software*.
- Infraestrutura de segurança: Conjunto de membros responsáveis pela deteção de falhas de segurança e vulnerabilidades. Periodicamente fazem verificação de *software* em cada centro individualmente. Cada centro é escolhido de forma aleatória.
- Serviços *core*: Serviço responsável pela gestão dos principais serviços *grid*, como gestão de utilizadores numa VO, BDII (Berkeley Database Information Index) de topo, WMS (Workload Management System), catálogo de ficheiros central entre outros.

2.5.2 Serviços Técnicos

Este serviço tem como objetivo melhorar o uso de utilização da infraestrutura e de dar suporte técnico à comunidade. Eis então, alguns serviços:

- Serviços organização virtual: Serviço responsável por dar informação a toda a comunidade, tal como, documentação, ferramentas existentes, serviços.
- Serviços de *software*: Serviço responsável pelo repositório central de *software*. Todo o *software* submetido para este repositório é validado, pois este tem de passar por um conjunto de critérios de qualidade antes de poder ser colocado em *staged rollout*. Esses critérios são definidos pela *Technology Coordination Board* (http://go.egi.eu/QualityCriteria-1).
- Base de dados de aplicações – AppDB: É uma base de dados que guarda um conjunto de aplicações sobre os mais diversos campos da ciência. Isto é útil para que os cientistas não tenham que despender tempo na realização de seu próprio *software* e também é útil para aqueles que não têm conhecimentos de programação.

A Figura 2.5 ilustra as diferentes áreas de investigação existentes e o seu respetivo uso, utilizando a infraestrutura EGI.

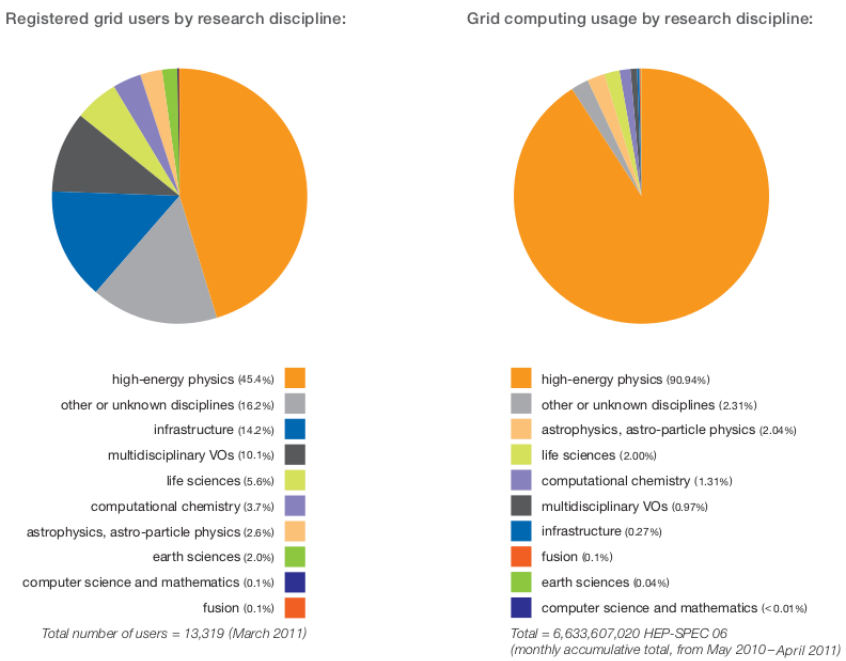

Figura 2.5: Áreas de investigação [88].

2.5.3 Recursos EGI

A Figura 2.6 mostra os recursos disponíveis na infraestrutura EGI.

2.5.4 Exemplos de uso

A EGI é utilizada para a execução de várias aplicações em domínios tão distintos quanto Medicina e Engenharia. Nesta secção apresentamos dois exemplos práticos:

- Diagnóstico da doença de Alzheimer: Como é conhecido existem milhões de pessoas que detêm esta doença e a tendência é de cada vez mais as pessoas a deterem à medida que o tempo vai passando. Esta doença é responsável por degradar o cérebro e por isso as pessoas têm perda de memória dos acontecimentos recentes e também limita a atividade física conforme a doença vai agravando. A chave para o tratamento desta doença é a deteção da mesma numa fase inicial. Então para que isto possa ser possível as imagens do cérebro do paciente precisam de ser comparadas com imagens de cérebros de outras pessoas saudáveis. É aqui, nesta fase, que a infraestrutura *grid* entra em ação, esta é responsável pela comparação das diversas imagens e de dar o diagnóstico. Quanto mais imagens forem comparadas maior será a precisão do resultado. Este processamento de imagens é bastante importante para

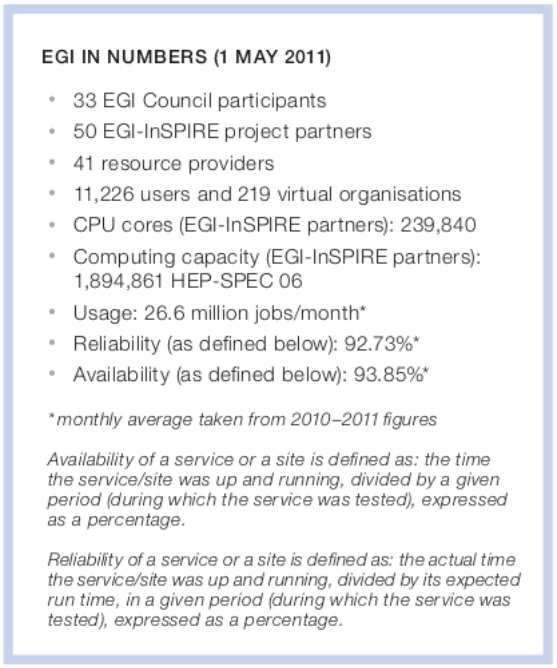

Figura 2.6: Recursos [88].

que os médicos consigam ver a progressão da doença ao longo do tempo.

- Tratamento de dados gerados pelo LHC (Large-Hadron Collider): Como é conhecido o LHC é o maior acelerador de partículas do mundo situado na fronteira entre a Suíça e França. Este gera quantidades enormes de dados anualmente que precisam de ser processados (cerca de 15 petabytes por ano). A infraestrutura EGI é uma das infraestruturas *grid* que mais contribui para este propósito. Esta contribui com biliões de horas de processamento todos os anos para o processamento de dados. A EGI é a parceira maior pertencente ao WLCG "Worldwide LHC Computing Grid", ou seja, esta infraestrutura é a mais utilizada pelo CERN (European Organization for Nuclear Research).

2.6 GLite

GLite é um *middleware grid* que foi desenvolvido pelo projeto EGEE "Enabling Grids for E-science". Este foi criado graças ao esforço conjunto de vários cientistas e engenheiros por todo o mundo [85]. O objetivo principal deste *middleware* é o de fornecer aos cientistas um conjunto de recursos computacionais geograficamente distribuídos, onde estes possam tirar proveito das grandes capacidades de processamento e armazenamento cooperativo [70]. Este *middleware* permite a cooperação de várias organizações científicas, trazendo assim avanço tecnológico mais rápido e sustentável. O gLite é construido tendo como base um *toolkit* criado na Universidade de Chicago com a colaboração do ANL (Argonne

National Laboratory) e globus [75], que, por sua vez, se baseia na arquitetura OGSA.

Como dito na secção 2.3 todos os utilizadores são agrupados em organizações virtuais (VO), que por sua vez cada VO é um grupo de trabalho ou organização. Segue-se então agora, uma breve explicação do funcionamento interno do *middleware* gLite, isto para dar uma perceção mais clara sobre o seu funcionamento.

2.6.1 *Information Service*

O sistema de informação é essencial para fornecer informação de todos os recursos existentes e os seus diferentes estados de operação. Esta informação é crucial para todas as operações *grid*, porque é através deste serviço que os recursos são descobertos e monitorizados [76]. Existem dois tipos de serviços de informação no gLite: R-GMA e MDS. R-GMA (Relation-Grid Monitoring Architecture) é uma base de dados relacional que serve essencialmente para fazer procura de informação. Essa informação, normalmente, é sobre os utilizadores e sobre a monitorização dos recursos/tarefas. A informação é produzida por produtores (mudança de estado de uma tarefa) e é consumida por consumidores (p.e. utilizadores). Toda a informação produzida é encaminhada para um registo central que por sua vez serve de canal de comunicação entre produtores e consumidores [2]. Esta base de dados contém a mesma informação que o servidor de informação BDII (Berkeley Database Information Index Server). A diferença é que este não é usado pelo *Workload Manager* para escalonar a submissão de tarefas [85]. O MDS (Globus Monitoring and Discovering System) é uma implementação LDAP (Lightweight Directory Access Protocol) que contém todos os dados sobre todos os recursos *grid*. O MDS na prática é uma estrutura em árvore em que toda a informação sobre os recursos é recolhida em cada nível [70]. A arquitetura do sistema de informação pode ser vista na figura 2.7.

Cada BDII mais profunda (*resource-level* BDII) recolhe informação sobre os recursos (como por exemplo *computing e storage element*) e passa essa mesma informação à BDII imediatamente acima (*site-level* BDII) [84]. Posteriormente a *site-level* BDII vai passar toda a informação à BDII de topo. De notar que a BDII de topo contém toda a informação da infraestrutura e esta é essencial para a descoberta de recursos na *grid*.

2.6.2 *Workload Management System* (WMS)

O WMS é o núcleo de toda a infraestrutura. Este é responsável por receber todas as tarefas submetidas pelos utilizadores e atribui-las aos recursos computacionais mais apropriados. Este serviço é também responsável pela gestão de tarefas, tal como, verificação do estado da tarefa, cancelar tarefa e obter o output. O WMS está subdividido em várias componentes responsáveis por toda esta gestão. Então segue-se uma descrição dessas componentes integradas no WMS.

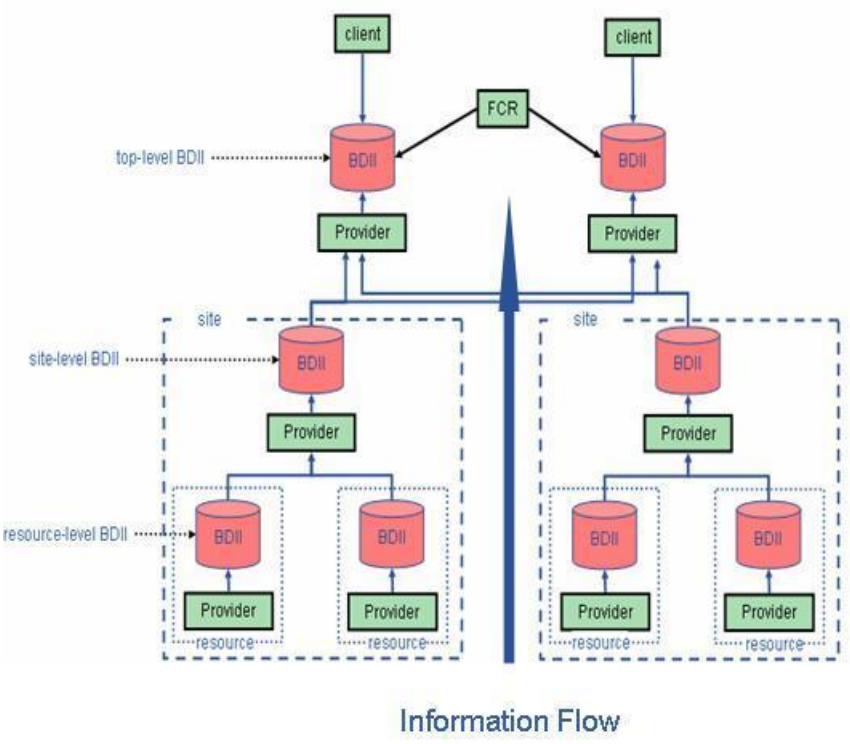

Figura 2.7: Serviço de informação [70].

2.6.2.1 *Workload Manager* (WM)

O WM é a componente principal do WMS. Este é responsável por satisfazer todos os pedidos dos utilizadores. Para satisfazer tais pedidos, este vai chamar outras componentes (descritas mais à frente) e em que a cooperação de todas irá permitir que cheguem a uma conclusão [64].

Um utilizador para submeter uma tarefa, terá que a definir numa linguagem chamada JDL (Job Description Language). Esta linguagem contém um conjunto de instruções que irão ser impostas pelo utilizador para executar determinada tarefa. Este serviço também é responsável pela monitorização de todas as tarefas submetidas [85].

2.6.2.2 *Task Queue*

É a componente responsável por guardar todos os pedidos efetuados por parte dos utilizadores. Por outras palavras, todos os pedidos irão ser guardados numa fila de espera para posteriormente serem tratadas pelo WM. Sempre que alguma tarefa não tenha recursos disponíveis, esta volta imediatamente para a fila de espera para tentar ser atribuída mais tarde [85].

2.6.2.3 *Matchmaker*

É a componente responsável por descobrir todos os recursos disponíveis que possam executar determinada tarefa. O processo de *matchmaking* é responsável pela solicitação de informação ao *Information Supermarket*, que contém a informação mais atualizada dos recursos. A informação pedida irá ter em atenção os requisitos que a tarefa necessita, ou seja, a informação irá vir filtrada [85]. Este processo de *matchmaking* é essencial para que o processo de escalonamento possa ser acelerado e que a tarefa seja atribuída aos melhores recursos.

2.6.2.4 *Information Supermarket*

Esta componente funciona como uma espécie de *cache*, necessária para acelerar o processo de *matchmaking*. Esta contém toda a informação dos recursos e é constantemente atualizada via *Information updater* [85].

2.6.2.5 *Information Updater*

Componente responsável por manter o *Information Supermarket* sempre atualizado. O processo de atualização é feito através da leitura da BDII de topo pertencente a uma organização virtual [85].

2.6.2.6 *Job Handler*

Job Handler é a componente responsável por toda a interação entre a tarefa e o *Computing Element* (CE). Quando o processo de *matchmaking* acaba, a tarefa irá ser escalonada e irá ser chamado um *job adapter* responsável por empacotar a tarefa que posteriormente será passada ao *CondorC*. Antes da submissão da tarefa para o CE um *job wrapper* cria todo o ambiente necessário no CE escolhido pelo escalonador. O *CondorC* é a componente responsável pela submissão e eliminação de tarefas. Depois de as tarefas serem submetidas, a componente *Log Monitor* irá ser responsável por monitorizar e registar todas as mudanças de estado das tarefas [85].

2.6.2.7 *WM-Proxy*

É o serviço responsável por dar acesso a todas as funcionalidades do WMS. Para um utilizador poder aceder à infraestrutura, primeiro deverá ter um certificado válido devidamente assinado por uma autoridade certificadora válida. Posteriormente deverá separar a sua chave privada e pública do certificado para que possa assinar um pedido de delegação de *proxy*. O *WM-proxy* irá ser contactado para criar um certificado *proxy* (certificado temporário). Este é responsável por passar um conjunto de atributos válidos a esse tal certificado temporário. O certificado *proxy* é uma delegação de direitos feitos pelo

utilizador para que o identifique durante um período de tempo (normalmente 12 horas) dentro da infraestrutura. Este *proxy* irá ter as permissões necessárias para que o sistema possa executar um conjunto de tarefas em nome do utilizador. Os certificados das autoridades certificadoras poderão ser consultados/descarregados em `http://www.igtf.net/`. IGTF (International Grid Trust Federation) é uma comunidade responsável por estabelecer políticas e diretrizes comuns entre PMAs (Policy Management Authorities). PMAs são organizações que coordenam autenticações confiáveis [25]. Por exemplo, a EUGridPMA é responsável por coordenar a autenticação na Europa para e-Science [13].

2.6.3 *Computing Element* (CE)

Um CE é um conjunto de recursos computacionais (*clusters*) localizados em algum lugar, que posteriormente vão ser responsáveis pela execução das tarefas submetidas [70]. Um CE é composto por três elementos básicos [85]:

- *Grid Gate* (GG): Interface de comunicação entre a *grid* e o *cluster*. Este é responsável pelo controlo de todas as operações de entrada e saída das tarefas dentro do CE.
- *Local Resource Management System* (LRMS): Este serviço é responsável pela alocação de recursos no *cluster* e é também responsável por guardar todas as tarefas numa fila de espera para uma posterior execução.
- *Worker Nodes* (WN): Tal com o nome indica, estas são as máquinas propriamente ditas que vão executar as tarefas, ou seja, são as máquinas físicas concentradas num local.

Para finalizar, cada CE poderá ter diferentes filas e cada fila corresponde a um conjunto de *Workers Nodes* distintos.

2.6.4 Gestão de Dados

A gestão de dados é essencial para a infraestrutura *grid*, pois esta permite gerir uma quantidade enorme de dados que muitas vezes são necessários para as tarefas serem executadas. A gestão de dados do gLite pode ser agrupada em três categorias. Segue-se então a descrição dessas mesmas categorias.

2.6.4.1 *Storage Element* (SE)

Um SE essencialmente serve para guardar dados, ou seja, este é um dispositivo de armazenamento onde são guardados os mais variados tipos de dados. Em gLite os ficheiros poderão ser replicados em diferentes SEs, para poderem fornecer acesso a diferentes

aplicações em diferentes localizações de forma eficiente. Existe no entanto uma limitação, limitação essa que só permite que os dados sejam lidos e não modificados ou escritos. Isto é necessário para garantir a consistência nos dados, ou seja, os dados só serão escritos durante a sua criação [85]. Como é normal ao existirem várias réplicas de ficheiros em localizações diferentes, o processo de sincronização entre eles era algo muito custoso, por isso a opção de criar ficheiros só *read-only*. Para colmatar esse problema existe uma API chamada GFAL que permite a um programa guardar e registar um ficheiro num SE. Sempre que o utilizador desejar escrever para um SE deverá usar esta API.

O SE é gerido por um gestor chamado *Storage Resource Manager* (SRM). O SRM é o *middleware* responsável por um conjunto de operações sobre dados entre diferentes SEs, de forma transparente para o utilizador. Essas operações poderão ser, transferências de ficheiros, reserva de espaço, re-nomeação dum ficheiro entre outras operações.

2.6.4.2 *Logical File Catalog* (LFC)

O LFC é um catálogo de ficheiros onde todos eles e as suas correspondentes réplicas são registados. Esse registo serve de identificação para um ficheiro, pois sem esse registo é impossível localizar os ficheiros. A identificação pode ser feita por quatro tipos de identificadores diferentes, vamos então mostrar os tipos de identificadores:

- GUID : é um identificador que identifica o ficheiro de forma única. O identificador é uma *string* de caracteres gerados através de uma combinação do endereço MAC e do *timestamp*, isto para assegurar que os identificadores sejam únicos. (p.e. guid:135b7b23-4a6a-11d7-87e7-9d101f8c8b70).
- LFN: é um identificador tal como o anterior, mas a diferença é que a *string* é humanamente legível. Isto facilita a interação com os ficheiros na *grid*. Cada ficheiro na *grid* (*grid file*) pode ter vários nomes apontados a ele (*links*), como acontece no sistema operativo UNIX. (p.e. lfn:myFile).
- SURL: este identificador contém informação sobre um ficheiro. A informação que contém é o endereço físico, o SE ou SRM e o caminho do ficheiro no SE. (p.e srm://gb-se-ams.els.sara.nl/dpm/els.sara.nl/home/lsgrid/generated/2008-02-28/file2868595b-ae21-4d6a-8a60-0ed4a668b809).
- TURL: este identificador fornece a informação sobre a localização do ficheiro, tal como a informação de acesso e como obtê-lo. Este também contém a informação sobre o protocolo de acesso e a porta à escuta. Cada ficheiro poderá ter vários TURLs, que correspondem aos diferentes protocolos suportados pelo SRM. Os identificadores GUID e LFN são usados unicamente para identificar ficheiros já os SURL e TURL são usados para obter informação dos ficheiros. (p.e. gsiftp://tbed0101.cern.ch/flatfiles/SE00/dteam/generated/2004-02-26/file3596e86f-11d7-a6b0-f53ee5a37e1d).

Por fim, estes identificadores são guardados num catálogo de ficheiros (LFC) onde são mapeados com os ficheiros guardados nos SEs. Sempre que um utilizador/programa

desejar aceder a um ficheiro, estes deverão primeiro consultar o LFC para poderem ter a referencia para o ficheiro desejado.

2.6.4.3 *File Transfer Service* (FTS)

FTS é um serviço gLite responsável por toda a gestão de ficheiros. Este é responsável por aceitar, escalonar, e executar transferências de ficheiros entre diferentes SEs. Este serviço no fim de submeter um ficheiro gera um ID e toda a informação que o identifica, para posteriormente este poder ser acedido por aplicações.

2.6.5 *Logging and Book-Keeping* (LB)

O serviço LB é responsável por rastrear uma tarefa. Esse rastreio é feito em termos de eventos, sempre que há alterações de estado a informação é publicada. Sempre que existe uma alteração de estado, essa é guardada num ficheiro local. Este por sua vez publica essa alteração a um *local logger* que por sua vez publica as alterações a um *inter-logger* e este faz chegar toda a informação a um servidor *book-keeping*. Sempre que um utilizador fizer um pedido de verificação de estado de uma tarefa, o estado a ser retornado é o que estiver no servidor *book-keeping*. Toda a informação de *logging* é assíncrona, isto para tentar reduzir ao máximo a carga pela rede [34]. Podemos dizer que *logging* é um dos problemas reais em *grid* porque muitas vezes a tarefa já acabou de correr a algum tempo e o estado no servidor *book-keeping* ainda não foi alterado. As alterações de informação demoram tempo a ficarem consistentes e este problema e um dos que faz com que a execução de uma tarefa seja morosa em *grid*.

O serviço LB tem também a possibilidade de notificar os seus utilizadores (se o desejarem), quando algumas mudanças em particular acontecem. Essa informação depois de chegar ao servidor *book-keeping* irá ser publicada (produtor) a um sistema de informação que por sua vez será responsável por entregar ao utilizador final, como pode ser visto na Figura 2.8. De notar que a infraestrutura utiliza o sistema de informação R-GMA para fazer chegar a informação ao utilizador. (para mais informação consultar a subsecção Sistema de Informação deste capítulo) [85].

2.6.6 *User Interface* (UI)

A UI não é nada mais nada menos que o ponto de acesso entre o mundo exterior e a infraestrutura *grid*. UI é uma máquina, onde os utilizadores têm uma conta pessoal, um certificado pessoal e um conjunto de comandos disponíveis para a interação com a *grid*. Este serviço oferece um conjunto de comandos CLI (Command Line Interface) que permite aos utilizadores executarem algumas operações, tais como [70]:

- Autenticação na *grid*, que por sua vez lhe vai dar acesso a um conjunto de recursos (autorização) dependendo da organização virtual a que pertença.

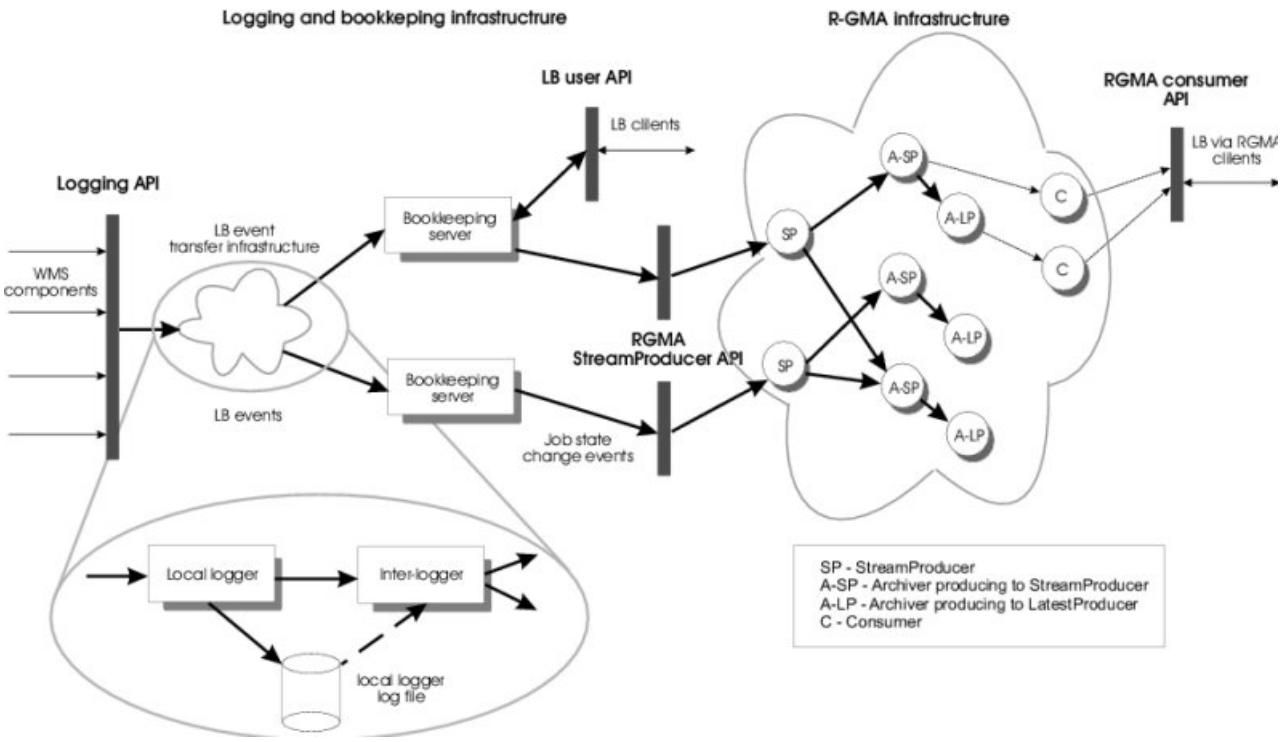

Figura 2.8: Infraestrutura *logging and book-keeping* [34].

- Listar todos os recursos disponíveis para uma determinada tarefa.
- Submeter tarefas para execução e cancelar tarefas.
- Perguntar sobre o estado de execução e transferir o output da mesma.
- Copiar, replicar e apagar ficheiros na grid.
- Obter o estado de diferentes recursos a partir do sistema de informação.
- Utilizar um conjunto de APIs disponíveis para interação com a infraestrutura.

Além disso, esta também fornece APIs de acesso a vários serviços o que permite desenvolver aplicações *Grid-Enable*, isto para automatizar alguns processos. É claro que para usar as APIs os utilizadores precisam ter conhecimentos de programação.

2.6.7 *Virtual Organization Membership Service* (VOMS)

O serviço VOMS é utilizado como repositório central de informação sobre os utilizadores. Essa informação é sobre a autorização que os utilizadores detêm (utilizadores pertencentes a uma VO). Basicamente este serviço é responsável por definir regras sobre um utilizador. O VOMS está subdividido em três partes principais [20].

- Uma base de dados Mysql que é um repositório persistente de toda a informação sobre os membros pertencentes a determinada VO.
- VOMS *admin*, que é uma interface *web* disponível em que o administrador é o responsável por atribuir permissões a todos membros. Essa interface *web* corre sobre Apache/Tomcat.
- VOMS *server*, que é um servidor responsável por criar o certificado *proxy* de um utilizador, dando assim as permissões definidas em VOMS *admin*.

2.6.8 APEL

APEL é uma ferramenta de contabilidade que permite gerar estatísticas para posteriormente serem analisadas. Esta ferramenta coleciona informação sobra a infraestrutura. Toda a informação é então servida numa base de dados central, para posterior geração de estatísticas [5].

2.6.9 *Job Description Language* (JDL)

JDL é uma linguagem de alto nível usada para especificar tarefas usando um conjunto predefinido de parâmetros. Esta linguagem é usada pelo WMS para "compreender" a especificação de uma tarefa e levantar os requisitos necessários. Com base nesses requisitos irão ser atribuídos os melhores recursos disponíveis que satisfaçam os requisitos pretendidos.

A criação de um ficheiro JDL é feito essencialmente com linhas com o seguinte formato "*atribute=expression*". As expressões podem ter tamanho variável e todas terminam com ponto e virgula. Os comentários são precedidos pelo carácter "//" ou "#" e para comentar várias linhas pode-se usar "/*" e "*/". Esta linguagem é muito sensível aos espaços em branco, nos atributos, expressões e/ou depois do ponto e virgula. Segue-se então um pequeno exemplo que mostra a definição de uma tarefa simples:

```
1    Type          = ``Job'';
2    JobType       = ``Normal'';
3    Executable    = ``/bin/hostanme'';
4    Arguments     = ``-f'';
5    StdOutput     = ``file.out'';
6    StdError      = ``file.err'';
7    OutputSandBox = {``file.out'', ``file.err''};
8    RetryCount    = 7;
```

Figura 2.9: Exemplo JDL

O valor do atributo *Type* é sempre *Job* e existe por razões históricas. O atributo *JobType*

especifica o estilo da tarefa. O estilo de uma tarefa poderá ser diferente, como por exemplo *parametric*, *MPICH*, *Interactive* etc. O atributo *Executable* especifica o executável/-comando a correr num *computing element*. O atributo *Arguments* são os argumentos necessários para o executável/comando. O atributo *StdOutput* define o ficheiro onde vai ser escrito o standard de *output* (file.out), o mesmo para *StdError* mas o standard é o de erro (file.err). O atributo *OutputSandBox* especifica o conjunto de ficheiros a serem transferidos quando o executável/comando acabar a execução. Por fim o atributo *RetryCount* permite definir o número de tentativas de resubmissão da tarefa em caso de falha.

Os atributos descritos acima são os mais utilizados no *middleware* gLite, porém existem outros mais mas que não são tão relevantes para o objetivo desta dissertação e também para o utilizador comum. Para finalizar este capítulo, é importante falar sobre segurança em gLite, visto ser um tópico de extrema importância para que possa existir uma infraestrutura segura e fiável.

2.6.10 Segurança

Como visto anteriormente toda a comunidade *grid* é dividida em organizações virtuais (VOs). Isto é feito para diferenciar grupos de trabalho para que todos os elementos pertencentes a um grupo colaborem entre todos para chegarem a um determinado objetivo comum. A camada de segurança GSI "Grid Security Infrastruture" oferece um nível de segurança a toda a infraestrutura, como autenticação, autorização, comunicações seguras etc. GSI é baseado em encriptação de chave pública, certificados X.509 e a comunicação é feita usando o protocolo SSL (Secure Socket Layer) [70].

Um utilizador que deseje utilizar os recursos *grid*, primeiro terá de pertencer a uma organização e registar-se numa VO. Depois disso deverá pedir um certificado pessoal a uma autoridade certificadora reconhecida pela infraestrutura. Quando o utilizador tiver o certificado em sua posse, deverá separar a chave privada e pública e colocá-las num diretório *.globus* na UI. A chave privada é protegida por uma palavra-chave e esta é essencial para gerar o certificado temporário, chamado *proxy certificate*. Ao criar esse certificado temporário o utilizador está a delegar os seus direitos a este. O certificado *proxy* vai então servir como meio de autenticação e autorização em todos os recursos *grid* sem intervenção humana (*Single sign-on*). A criação deste certificado é feita através da assinatura da chave privada de um utilizador e também através da ligação ao servidor VOMS que lhe vai facultar os direitos do utilizador. O certificado temporário terá uma duração, por defeito, de 12 horas. Isto para reduzir o risco de segurança do certificado em caso de roubo [70].

Para finalizar, quando um utilizador criar o certificado temporário, este terá acesso a todos os recursos, pertencentes a sua VO, durante um determinado período de tempo sem ter-se de autenticar novamente. Para tarefas de longa duração ($>$ 12horas), os utilizadores terão de criar um servidor *MyProxy* para que o certificado possa ser renovado enquanto as tarefas estiverem em execução.

2.7 Tipos de Tarefas e Estados Durante o Seu Ciclo de Vida

O tipo de tarefas que normalmente podem ser submetidas para a infraestrutura *grid* são: *Normal*, *Parametric*, *Collection*, DAG (Direct Acyclic Graphs) e MPI (Message Passing Interface). Uma tarefa *Normal* é uma tarefa única em que é executado somente um programa/executável. Por exemplo, correr o programa *hostname*, que fornece o nome da máquina. Uma tarefa *Parametric* é uma conjunto de tarefas que são submetidas para a infraestrutura *grid* como fosse uma só. Esta é muito usada para correr tarefas muito parecidas mas com argumentos diferentes, permitindo poupar tempo ao utilizador na definição de diferentes tarefas. É atribuído um *id* a cada sub-tarefa para a poder gerir individualmente, mas também é atribuído um *id* a todo o conjunto de tarefas para que estas possam ser geridas como um grupo [42]. Uma tarefa do tipo *Collection*, é um conjunto de tarefas diferentes que são submetidas e geridas pelo utilizador como se trata-se de um tarefa só [32]. Uma tarefa do tipo DAG, é uma forma de submeter tarefas que estão relacionadas e que dependem dos resultados umas das outras. Ou seja, este tipo de tarefa permite criar dependências entre tarefas em que as tarefas dependentes só começam a execução quando as suas dependências forem resolvidas. Por exemplo, uma tarefa B só começa a execução quando a tarefa A terminar, porque B necessita dos resultados calculados em A [9]. Por fim, uma tarefa do tipo MPI, é uma tarefa que irá ser executada em paralelo na infraestrutura *grid*. Estes tipos de tarefas irão ser submetidas para ambientes distribuídos, onde posteriormente serão executados [35]. Neste caso um ambiente distribuído é um ambiente que suporta a execução de programas MPI.

Quando um utilizador submete uma tarefa para a *grid*, esta passa por um conjunto de diferentes estados de execução. A figura que se segue mostra todas as possibilidades de estados e transições a que uma tarefa está sujeita. De notar que se o tipo de tarefa for DAG (Directed Acyclic Graph), esta não passará pelos estados *READY* e *SCHEDULED* [70].

Na Figura 2.10 podem ser vistas todas as transições possíveis de estados [70]:

- *SUBMITTED*: A tarefa foi submetida com sucesso (sintaxe do ficheiro JDL foi verificada a aceite) pelo utilizador, mas ainda não foi processada pelo WMProxy (responsável pela autenticação e permissões da tarefa).
- *WAITING*: A tarefa já foi aceita pelo *WM-Proxy*, mas ainda não foi passada ao WM, ou seja, a tarefa autentica-se com sucesso detendo assim as permissões necessárias.
- *READY*: A tarefa já foi atribuída a um CE, mas ainda não foi transferida para a sua fila de execução.
- *SCHEDULED*: A tarefa está à espera de ser executada na fila do CE atribuído.
- *RUNNING*: A tarefa está no momento em execução.
- *CANCELED*: A tarefa foi cancelada pelo utilizador.

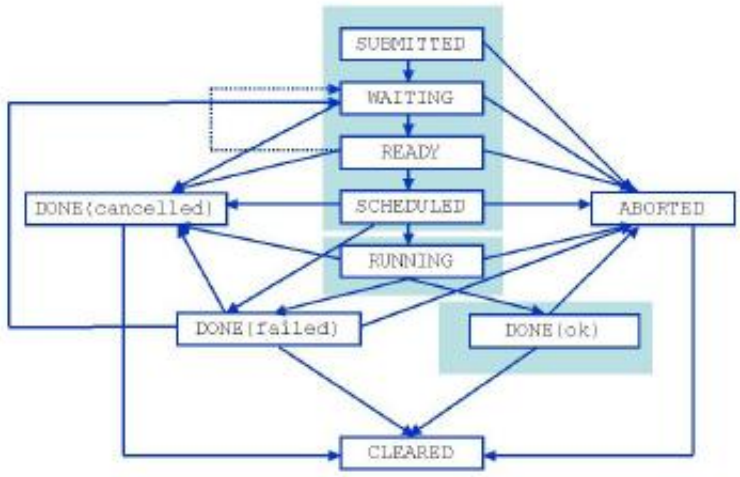

Figura 2.10: Estados possíveis de uma tarefa [70].

- *CLEARED*: O utilizador fez o pedido de transferência do atributo *OutputSandBox*. Os ficheiros são transferidos para a UI e todo a ambiente alocado para a tarefa é eliminado. De notar que uma vez que os ficheiros do *OuputSandBox* sejam transferidos para a UI, esses mesmos ficheiros no CE serão imediatamente eliminados não havendo forma de reavê-los novamente.

- *DONE*: A tarefa terminou a sua execução. Tal como um programa normal, executado na nossa máquina pessoal, este pode terminar com sucesso ou não. O não sucesso na execução poderá ser, por exemplo, falta de ficheiros de entrada ou o ambiente onde está inserido não é o mais apropriado.

- *ABORTED*: A tarefa foi abortada pelo WMS (p.e. porque o certificado *proxy* expirou).

Uma tarefa ao ser submetida com sucesso não é sinónimo de ter sucesso na execução. Existe um conjunto de fatores que podem influenciar esse não sucesso na execução. Sempre que uma tarefa tem um erro, este será escrito no ficheiro de standard de erro definido no ficheiro JDL, para posteriormente ser analisado pelo utilizador. Um dos erros frequentemente cometidos é, submeter um programa para execução e não especificar o tipo de arquitetura onde este foi compilado (Se um programa foi compilado em X86, este não irá certamente executar numa arquitetura AMD). O utilizador tem duas soluções para este problema, especificar um *Script Shell* (SH) que compile e execute o programa na máquina remota. A outra solução é especificar o tipo de arquitetura pretendida no atributo *Requirements* no ficheiro JDL. A solução do *Script Shell* é muitas vezes adotada pelos utilizadores, pois estes podem adaptar todo o sistema à sua vontade, como por exemplo definir variáveis de ambiente.

2.8 Conclusão

Este capítulo centrou-se principalmente na descrição da arquitetura *grid* OGSA definida pelo OGF, pela descrição de *middleware* gLite e pela definição de *grid*. A infraestrutura *grid* oferece todo um conjunto de recursos computacionais, dispersos geográficamente, ao utilizadar. Toda esta infraestrutura é gerida recorrendo a organizações virtuais. Uma organização virtual define todo um conjunto de indivíduos e recursos, baseando-se em regras.

A Universidade do Porto disponibiliza recursos computacionais à infraestrutura de *grid* europeia EGI. EGI é uma infraestrutura composta por diversos países da União Europeia, e todos eles têm funções específicas de gestão e administração que fazem com que toda esta infraestrutura se consiga manter operacional. O *middleware* utilizado pela UP chama-se gLite. O *middleware* gLite permite gerir todos os recursos dispersos, por forma a oferecer toda uma infraestrutura de computação e armazenamento a todos os utilizadores. Os utilizadores para utilizarem a infraestrutura existente terão que criar a sua tarefa numa linguagem chamada JDL, que posteriormente será submetida para a infraestrutura. Essa tarefa irá passar por um conjunto de diferentes estados que vão desde a sua submissão à sua conclusão.

Capítulo 3

Portais *Grid*

3.1 Introdução

Como mencionado anteriormente, computação *grid* é uma tecnologia que permite a todos os utilizadores, registados, utilizar um conjunto de recursos computacionais para resolverem determinado problema. Todos esses recursos estão distribuídos geograficamente e permitem executar todo um conjunto de aplicações de forma distribuída e paralela. A utilização destes recursos normalmente é feita através de interfaces de linhas de comandos, o que forma a tarefa muito árdua para o utilizador. Para minimizar este problema, foram criados os portais *grid*.

Um portal *grid* é um ponto de acesso *web* que permite a todos acederem a uma grande variedade de recursos. Este fornece um ponto de acesso único aos recursos *grid*, desde que se tenha a autorização devida. Os portais são importantes para que todos os cientistas/engenheiros se concentrem somente na sua pesquisa deixando muitos detalhes de baixo nível para o portal e *middleware grid* [97].

Neste capítulo, mostramos o progresso no desenvolvimento de portais *grid* ao longo dos anos, tal como as tecnologias mais relevantes em cada época. Falamos também, de forma breve, sobre alguns portais *grid* existentes e enumeramos as suas vantagens e desvantagens. Este capítulo está organizado da seguinte maneira. A Secção 3.2, descreve as tecnologias utilizadas na primeira geração de portais *grid*, bem como a sua arquitetura e alguns serviços oferecidos. Nesta descrevemos também as suas limitações que levaram à criação da geração seguinte. Na Secção 3.3, apresentaremos a segunda geração de portais *grid*, bem como a tecnologia que marcou essa geração. A tecnologia é chamada de *portlet* e esta secção concentra-se essencialmente nesta tecnologia. Para finalizar, na Secção 3.4 discutimos sobre os diferentes portais *grid* existentes, as suas arquiteturas, vantagens e desvantagens.

3.2 Primeira Geração de Portais *Grid*

A primeira geração de portais *grid* consistia em páginas *web* tradicionais, onde o utilizador podia encontrar todo um conjunto de serviços que naquela época eram inovadores. Esses serviços permitiam ao utilizador, realizar todo um conjunto de serviços que habitualmente eram feitos somente diretamente no seu sistema operativo [84]. Os serviços oferecidos eram os seguintes:

- Autenticação: Cada utilizador autentica-se no portal detendo assim as permissões necessárias de acesso a todo um conjunto de serviços oferecidos pelo portal. Esta autenticação permite ao utilizador delegar direitos aos serviços. Sendo assim, os serviços passam a ter permissões para executar funções que só o utilizador em questão poderia realizar [84].
- Gestão de tarefas: O portal oferece ao utilizador a capacidade de definir todo um conjunto de tarefas para que estas possam ser executadas de forma confiável e segura. O utilizador tem também a possibilidade de verificar os diferentes estados das tarefas submetidas, podendo inclusive cancelá-las e descarregar os resultados gerados para a sua máquina local [84].
- Transferência de dados: O portal permite que os utilizadores submetam dados (ficheiros) necessários para a execução da sua tarefa. Esses dados poderão ser, por exemplo, ficheiros de entrada (*input*) de um programa/executável. Posteriormente, após o término da execução de uma tarefa, o utilizador poderá descarregar todos os resultados gerados para a sua máquina local. Os utilizadores têm também a possibilidade de guardar ficheiros em diferentes localizações da infraestrutura *grid* bem com todos os seus resultados, se assim o desejarem [84].
- Serviços de Informação: Este serviço é responsável por descobrir toda a informação sobre todos os recursos *grid*. Essa informação é posteriormente analisada por um escalonador, responsável por alocar os melhores recursos para determinada tarefa. A alocação dos recursos é feita com base nos requisitos e prioridades que o utilizador define, quando cria a sua tarefa. Os serviços de informação além de servirem como base ao escalonador, também permitem que todos os utilizadores façam pedidos sobre o estado da infraestrutura. Sendo assim os utilizadores poderiam consultar tal informação neste tipo de portais [84].

Para que os serviços pudessem ser implementados, os portais de primeira geração apresentavam uma arquitetura que também não era comum nas páginas *web* tradicionais. Então a sua arquitetura poderia ser dividida em três camadas, sendo estas a camada interface, camada servidores *web* e camadas de serviços *backend* e de recursos [84], A Figura 3.1, ilustra a arquitetura desta primeira geração de portais e segue-se a descrição de cada camada:

- Camada *Interface* (Tier 1): Consiste num navegador *web*, onde qualquer utilizador (devidamente autenticado) pode definir um conjunto de instruções que posterior-

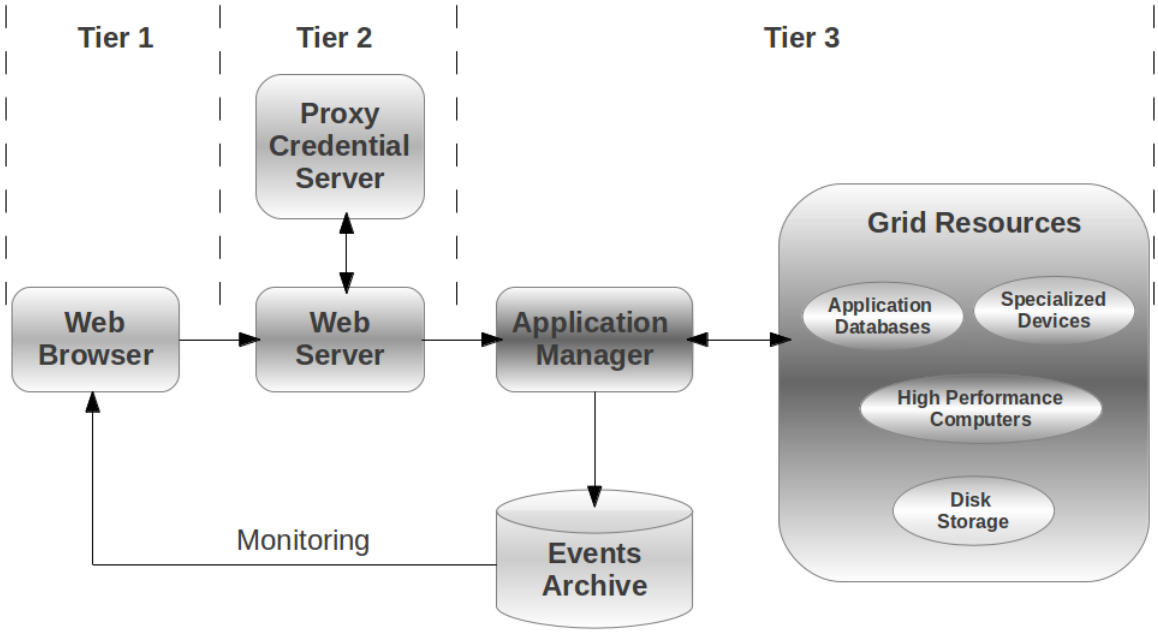

Figura 3.1: Primeira geração de portais, arquitetura [84].

mente irão ser executadas na infraestrutura *grid*. As instruções que podem ser executadas são por exemplo transferência de ficheiros, gestão de tarefas, autenticação, dentre outras [84].

- Camada *Servidores Web* (Tier 2): Esta camada serve de ponto de ligação entre o utilizador e a infraestrutura *grid*. É responsável pela autenticação do utilizador na infraestrutura, obtendo assim um certificado *proxy* válido a partir de um servidor *Proxy Credential*. O servidor *Proxy Credential* é responsável pela passagem dos parâmetros necessários à criação de um certificado *proxy*. Esses parâmetros são, normalmente, as permissões que os utilizadores detêm numa determinada organização virtual. O certificado *proxy* é a forma que o utilizador tem para delegar direitos à infraestrutura. Esses direitos servem para que os serviços atuem em nome do utilizador. A camada servidores *web* escuta todos os pedidos vindos dos utilizadores e trata-os devidamente. Sempre que um utilizador deseja executar alguma ação, esta camada responsabilizar-se-á por lançar um *Aplication Manager*. O *Aplication Manager* é um serviço responsável por toda a gestão de tarefas, controle e monitorização. Este serviço é uma espécie de consultor, em que o utilizador tem a possibilidade de consultar para verificar, por exemplo, os diferentes estados das tarefas. Para finalizar este ponto, este serviço irá ter as mesmas permissões que o utilizador teria na infraestrutura *grid*, pois essas serão delegadas por ele, delegando assim ao serviço um *credential proxy* [84].

- Camada Serviços *Backend* e de Recursos (Tier 3): Esta camada é a infraestrutura *grid* em si, onde irão ser executadas todas as tarefas submetidas por parte dos utilizadores. Esta irá ser também responsável por armazenar todos os ficheiros, que sejam necessários, e por todo o movimento destes que possa surgir. Para que tudo

acima descrito possa funcionar corretamente, toda a infraestrutura contém todo um conjunto de base de dados, *clusters*, vários dispositivos de armazenamento (*disk storages*), servidores de monitorização, entres outros [84].

Como referido anteriormente, os portais de primeira geração eram páginas *web* comuns, visto que utilizavam as mesmas tecnologias. As tecnologias utilizadas eram HyperText Markup Language (HTML), Java Server Pages (JSP), Javascript, Common Gateway Interface (CGI), Perl, entre outras. É dada a seguir uma breve descrição sobre a função de cada uma destas tecnologias:

- HTML: Linguagem de marcação. Linguagem definida através de elementos chamadas *tags* que permitem criar toda a estrutura de uma página *web*. Todos os elementos começam com uma *tag* de abertura, $<tag>$, e terminam com um *tag* de fecho, $</tag>$ [24].

- Javascript: É uma linguagem de *script* orientada a objetos que corre no lado do cliente, isto é, no navegador *web*. Esta linguagem permite aliviar a computação do lado do servidor fazendo com que alguma computação seja feita no lado do cliente, permitindo assim distribuir a carga de trabalho pelos vários cliente existentes (distribuição). Javascript é uma linguagem que complementa a linguagem HTML, pois esta permite gerir eventos despoletados pelos utilizadores, criar novas partes HTML dinamicamente, sem ter que recarregar a página completa, criar animações, entre muitas outras coisas. Esta linguagem permite ao programador gerir páginas dinamicamente e criar aplicações mais robustas e com um *look and feel* mais atraente como, por exemplo, efeitos gráficos [30].

- Perl: Esta é uma linguagem de *scripting*, utilizada em UNIX, que reúne todo um conjunto de potencialidades de outras linguagem como C, *shell scripting* (sh), AWK, and sed. Esta linguagem fornece um poderoso serviço de manipulação de texto, que faz com que esta seja popular principalmente em servidores HTTP, devido a sua capacidade de *parsing*. Além de servidores HTTP esta linguagem é bastante utilizada em computação gráfica, administração de sistemas, programação de rede, bioinformática etc [44].

- CGI: É um *script*, alojado num servidor *web*, que permite a troca de informação entre cliente e servidor. CGI permite eliminar a natureza estática das páginas HTML permitindo assim gerar páginas *web* dinamicamnete. Isto é possível porque no lado servidor um programa CGI é executado em tempo real, podendo assim mostrar ao cliente informação dinâmica - por exemplo, os resultados a uma pergunta, sobre determinado campo, a uma base de dados. Normalmente CGI é um módulo que é integrado em várias linguagens de programação tal como perl, python, PHP. Estas linguagens são aquelas mais utilizadas para servidores HTTP juntamente com a linguagem Java. Sendo assim, CGI permite aos utilizadores enviar pedidos, para obterem informação atualizada e dinâmica. Tanto CGI como Javascript geram páginas de conteúdo dinamicamente para o utilizador, mas a grande diferença é o local onde processam a mesma. Um *script* CGI é executado no lado do servidor,

já um programa Javascript é executado no lado do cliente [6]. Como é de esperar as duas tecnologias têm vantagens e desvantagens. Para a discussão sobre as vantagens e desvantagens das duas tecnologias consultar [7].

- JSP: É uma alternativa ao CGI para gerar conteúdos dinâmicos numa página web. JSP é uma tecnologia que utiliza a plataforma JavaEE para criar um servidor *web*. Esta tecnologia permite embeber Java juntamente com tags HTML e tags JSP. Cada página gerada com esta tecnologia juntamente com um servidor de aplicações, tal como, Tomcat [4] e Glassfish [17] é transformada em um *servlet*. Um *servlet* é uma classe em Java que processa dinamicamente todos os pedidos e respostas de um utilizador [31].

Para finalizar, os portais *grid* de primeira geração oferecem um conjunto de serviços básicos *task-oriented*, como autenticação, submissão de tarefas, transferência de dados e monitorização. Esta geração de portais permite que um utilizador comum possa submeter tarefas mais facilmente e rapidamente. Porém existem algumas limitações inerentes que a segunda geração compromete-se a resolver [84]:

- Falta de personalização: É impossível para os utilizadores personalizarem o portal à sua maneira respondendo de tal forma às suas necessidades, isto é, a capacidade de adicionar e remover alguns serviços [84].

- Serviços *grid* restritos: A primeira geração de portais são fortemente acoplados com o *middleware grid* que utiliza, tornando-se muito difícil de integrar diferentes serviços *grid* que requeiram outro tipo de *middleware* [84].

- Serviços *grid* estáticos: Como o ambiente *grid* é bastante dinâmico, com cada vez mais novos serviços, estes tornam-se cada vez mais difíceis de integrar nos portais de primeira geração devido à sua natureza estática, ou seja, incapacidade de gerir os serviços dinamicamente [84].

Para colmatar todos estes problemas as pessoas envolvidas na criação de portais *grid* adotaram novas abordagens dando origem à segunda geração de portais *grid*.

3.3 Segunda Geração de Portais *Grid*

A criação da segunda geração de portais *grid*, foi necessária para colmatar algumas limitações, descritas na secção anterior, da primeira geração. A solução adotada para resolver tais limitações foi a introdução de *portlets*, que permitem ao utilizador tratar várias secções da página de forma independente. Estes poderão ser adicionados e removidos dinamicamente em tempo de execução [84]. De seguida segue-se uma breve explicação sobre o que é um *portlet* e as suas respetivas características e benefícios.

3.3.1 *Portlets*

3.3.1.1 O que é um *Portlet*?

De uma maneira geral um *portlet* é uma janela num determinado portal ou página *web* que fornece um determinado serviço, como por exemplo notícias, calendário, isto tudo na perspetiva de um utilizador comum, Figura 3.2. Para um programador, um *portlet* é uma componente de *software* escrita em Java que gere todos os pedidos despoletados por utilizadores e que também é responsável por gerar conteúdo de forma dinâmica, como resposta aos pedidos oriundos dos cliente. Os *portlets* são componentes *pluggable* em que o utilizador tem a capacidade de adiciona-los, removê-los, modificar o seu *layout*, modificar toda a sua aparência, isto tudo de forma independente [84].

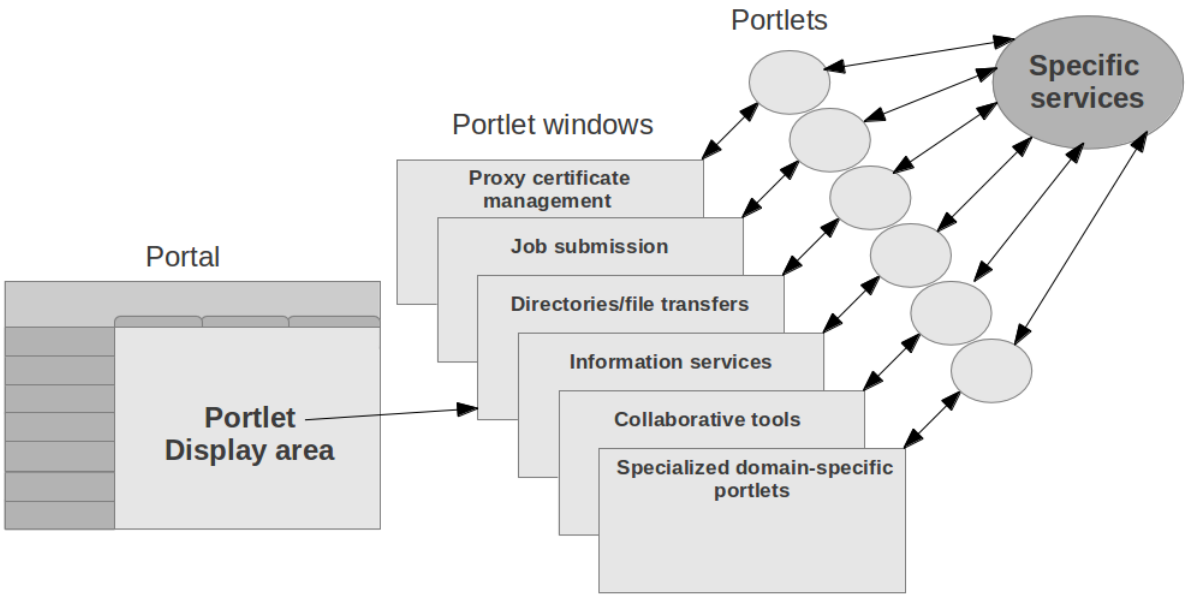

Figura 3.2: *Portlets* [97].

3.3.1.2 Serviços oferecidos por um *Portlet*

Uma das grandes potencialidades dos *portlets*, como dito anteriormente, é de fornecerem todo um conjunto de serviços que permite aos utilizadores ter uma interação gráfica gerada automaticamente podendo-a modificar/personalizar das mais variadas maneiras [59]. Eis alguns serviços que são comuns em *portlets*:

- *Single-sign on*: Como os *portlets* são janelas que funcionam independentemente, o utilizador necessita somente de se autenticar uma única vez no portal para ter acesso a todos os *portlets* disponíveis. De notar que se o utilizador adicionar um novo *portlet* e se este tiver uma sessão válida (de outra forma não conseguiria adicioná-los), o

portlet adicionado estará pronto a ser utilizado sem nenhuma preocupação, visto que o utilizador não necessita de se autenticar novamente (o utilizador deverá possuir as permissões necessárias para esta função, caso contrário, não a poderá concretizar) [59].

- Permissões: Para cada *portlet* existe um conjunto de permissões que podem ser modificadas de acordo com as necessidades, por exemplo poderá haver *portlets* que só determinados utilizadores podem visualizar [59] .

- *Pluggable*: Como dito anteriormente cada utilizador tem a capacidade para adicionar e remover *portlets* dinamicamente sem ter que reiniciar o servidor. Todas as alterações feitas a nível de programação são imediatamente visíveis ao utilizador, se assim for desejado [59].

- Personalização: O utilizador tem a capacidade de modificar todo o *display* gráfico de determinado *portlet*, isto de forma gráfica e todas as alterações ficarão guardadas a não ser que o *portlet* seja removido ou destruído [59].

Como é de esperar, existem outros tipos de serviços, mas estes enunciados acima são os principais. A tecnologia *portlet* é essencial para que os utilizadores possam adicionar/remover novos serviços de maneira dinâmica e fácil. Estes também são essenciais para que o portal possa ter diferentes conteúdos geridos de forma independente, mas que funcionem como um todo [59]. Para finalizar, uma das grandes vantagens em relação aos Java *servlets* ou aos servidores HTTP comuns é que o programador só tem que escrever uma só vez o código para diferentes dispositivos existentes no mercado. Por exemplo, os *portlets* são desenvolvidos para serem acedidos através de computadores pessoais, mas se estes forem acedidos através de qualquer dispositivo móvel, o servidor irá fazer a renderização automaticamente sem necessidade de qualquer intervenção [84]. Sendo assim, um portal que utilize a tecnologia *portlet*, poderá ser acedido pelos mais diversos dispositivos sem problemas de renderização.

3.3.1.3 Apresentação de um Portlet

A apresentação de uma janela *portlet* num portal ou numa página *web* comum consiste no seguinte [84], Figura 3.3:

- Uma barra titulo, com o título do *portlet*.

- Decorações, incluindo botões, para modificar o estado da janela de um *portlet*, como por exemplo maximizar, minimizar, modificar a aparência etc.

- Conteúdo produzido pelo *portlet*. Normalmente este conteúdo é gerado através de uma página JSP, em que o programador escreve código como se tratasse de uma página HTML tradicional mas com a possibilidade de inclusão de Java.

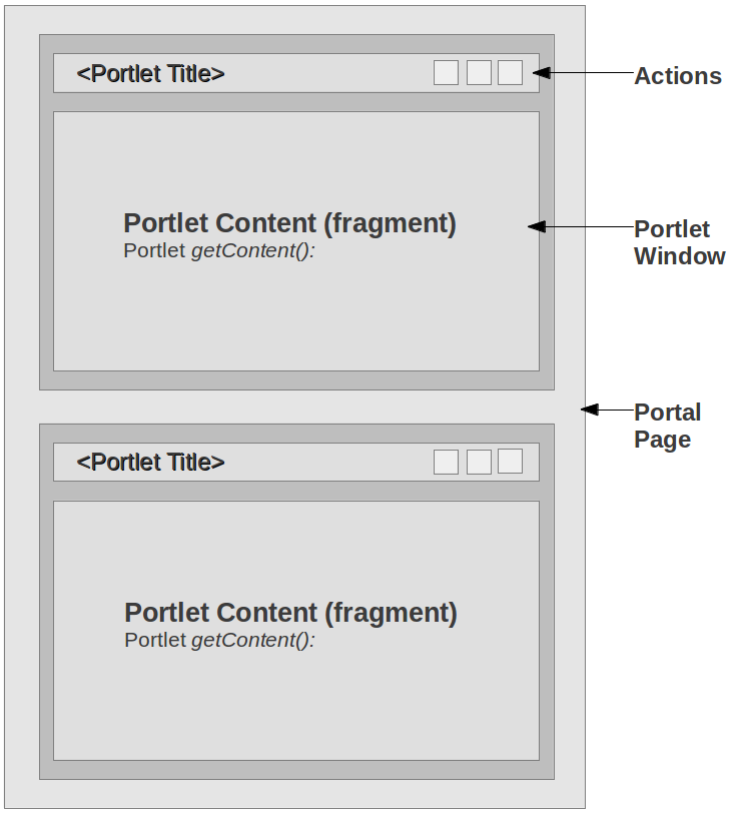

Figura 3.3: Apresentação de um *portlet* [84].

3.3.1.4 Ciclo de vida de um *Portlet*

Tal como páginas *web* dinâmicas, os *portlets* suportam diferentes estados, o que faz com que o seu ciclo de vida possa ser dividido em três partes principais:

- Inicialização: Nesta etapa, o *portlet* é iniciado com o conteúdo que o programador definiu no método *init()*. Este método serve essencialmente para inicializar o *portlet* quando este é mostrado no ecrã [97].
- Manipulação de pedidos: Esta é a fase em que o *portlet* está preparado para processar ações e renderizar conteúdos, oriundos de ações despoletadas pelos utilizadores [84]. Esta etapa pode ser sub-dividida em duas partes fundamentais, sendo estas:
  - Processamento de ações: Nesta fase o *portlet* vai processar todas as ações efetuadas pelos utilizadores, como eventos do rato, envio de formulários entre outros pedidos. O método *processAction()* vai estar à escuta de todos os pedidos e irá ter acesso a dois objetos *ActionRequest* e *ActionResponse* para processar pedidos e fornecer respostas, respetivamente. Os dois objetos irão ser passados como argumentos ao método *processAction()* [97].
  - Renderização de conteúdo: Esta fase serve essencialmente para mostrar os resultados de ações efetuadas no ecrã do requerente. O método *render()* é

responsável por renderizar todo o conteúdo necessário no ecrã do utilizador. Este método atualiza o *portlet* sem modificar o seu estado atual [97].

- Conclusão: Nesta fase o *portlet* irá chamar um método *destroy()* para remover o *portlet* do portal e para forçar a libertação de memória de toda alocação inerente [84] .

3.3.1.5 Especificações *Portlet*

Tal com acontece em muitas tecnologias no mundo da informática, foram criadas especificações standard para o desenvolvimento de *portlets*. No momento existem duas grandes organizações que definem esses padrões, e elas são, OASIS "Organization for the Advanced of Structured Information Standards" e JCP "Java Community Process". A definição de padrões é de extrema importância, pois só assim é possível que todos os *portlets* possam interagir uns com os outros sem grandes problemas. As especificações mais comuns são JSR 168 e JSR 286. JSRs "Java Specification Requests" são especificações *portlet* criadas pela organização JCP, que é responsável por definir todo um conjunto de Java APIs para que seja possível a interoperabilidade entre os portais e *portlets*. Esta especificação faz com que os *portlets* sejam *pluggable*, personalizáveis, e que incorporem mecanismos de segurança. A principal diferença entre as especificações JSR 168 e JSR 286 é que a segunda permite a comunicação entre diferentes *portlets*, enquanto a primeira não [84].

Para finalizar esta secção a segunda geração de portais *grid* veio solucionar os problemas existentes nos portais de primeira geração. Nesta geração, a introdução de *portlets* permite aos programadores que só se concentrem no trabalho em si e deixem os aspetos gráficos gerais para o portal. Com estes introduziu-se a capacidade da gestão de portais de forma completamente dinâmica [84]. Alguns benefícios que a segunda geração de portais acrescentou:

- Personalização do portal: Tal como dito anteriormente os utilizadores têm a capacidade de adicionar e remover *portlets* do portal e de personalização, isto tudo para ir de encontro às necessidades do utilizador.

- Serviços *Grid* Extensíveis: Os *portlets* são fracamente acoplados com o *middleware grid*, ou seja, o utilizador tem a capacidade de integrar serviços que funcionem em diferentes *middlewares grid*, bastando para esse efeito substituir o *portlet* que se adeque às suas necessidade.

- Serviços *Grid* dinâmicos: Os administradores, com esta tecnologia, têm a capacidade de publicar novos serviços sem interferir com o bom funcionamento do portal. Sendo assim, quando um serviço é publicado o utilizador final poderá utilizá-lo de imediato, desde que tenha as devidas permissões para adicioná-lo.

3.4 Portais *Grid* Existentes

Os portais *grid* oferecem um conjunto de ferramentas gráficas com que o utilizador pode contar, facilitando assim o uso da infraestrutura subjacente ocultando muitos detalhes de baixo nível.

Nesta secção serão descritos os vários portais *grid* existentes (os mais influentes), bem como a sua arquitetura, vantagens e desvantagens.

3.4.1 UCLA

UCLA é um portal *grid* onde os utilizadores devidamente registados podem submeter as suas tarefas. O portal *grid* UCLA juntamente com *UCLA Grid Architecture*, fornecem todo um conjunto de recursos computacionais que funcionam como uma infraestrutura *grid*. Os recursos de *hardware* que este utiliza são recursos computacionais (clusters) que a universidade de Califórnia, Los Angeles possui. A estrutura desses recursos computacionais consiste no seguinte [54]:

- Máquina *master*,
- Conjunto de clusters,
- Máquinas de armazenamento,
- Recursos de rede,
- *Software*,
- Recursos de dados.

O objetivo deste portal é oferecer a todos os utilizadores da instituição um meio de acesso fácil a todos os recursos disponíveis. Os utilizadores podem ter acesso a todos esses recursos através de uma única autenticação num portal *web*, submetendo todas as tarefas que desejarem. Os utilizadores acedem ao o portal *grid* através de uma sessão HTTPS o que lhes dá um canal de comunicação segura, tanto para o utilizador como para toda a infraestrutura que é importante salvaguardar. Todas as transações entre todos os recursos computacionais são feitas através de certificados X.509. A transação de certificados X.509 serve de autenticação para todos os recursos computacionais. De notar que o utilizador só necessita de se autenticar uma vez para ter acesso a diferentes tipos de recursos (*Single Sign-On*) [54].

O portal grid UCLA enquadra-se numa categoria diferente dos demais, pois este utiliza recursos internos próprios e a sua própria arquitetura. A arquitetura deste consiste numa interface *web* (portal), *Grid Appliance* e os recursos computacionais, Figura 3.4. O *Grid Appliance* serve de *gateway* entre o portal e todos os recursos computacionais da instituição. A instituição contém vários *Grid Appliance* o que faz com que tarefas sejam

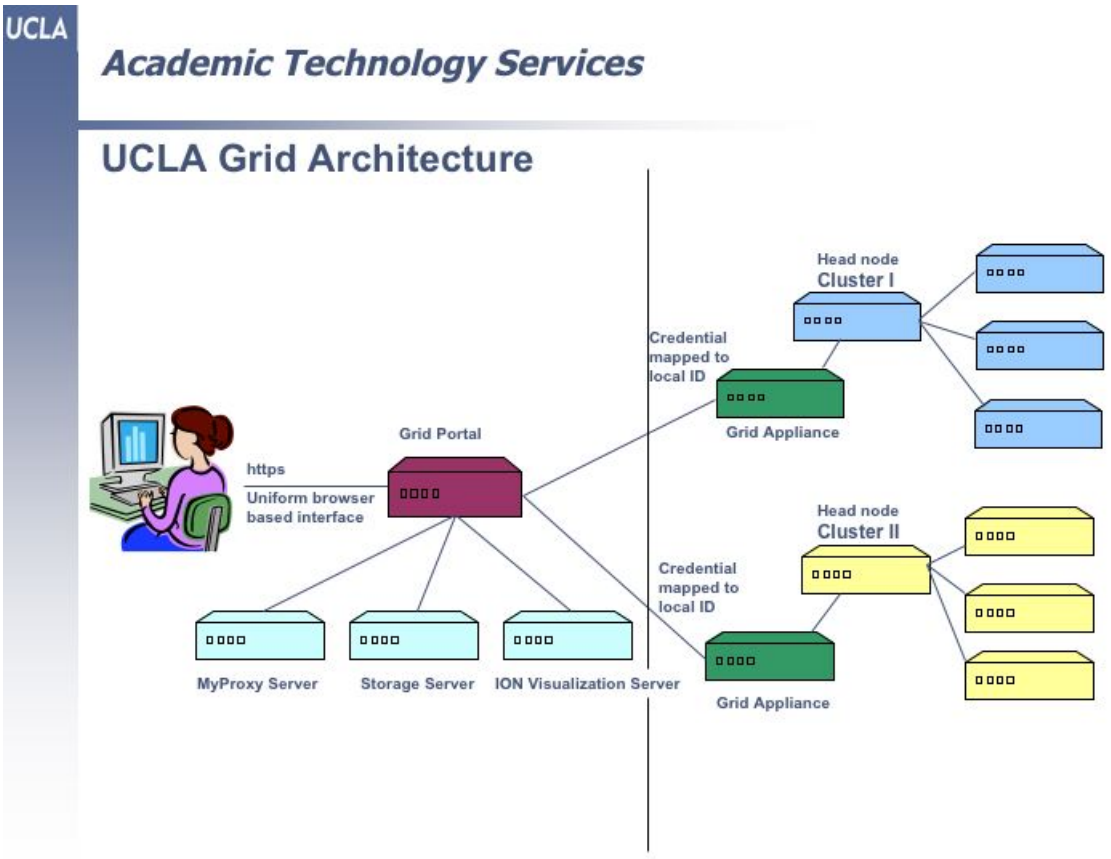

Figura 3.4: Arquitetura do portal UCLA [51].

distribuídas pelos diferentes *clusters* existentes [51]. Como é de esperar este portal não satisfaz as necessidades da Universidade do Porto, pois teríamos de mudar a arquitetura dos recursos computacionais. Para utilizar este tipo de arquitetura teríamos que colocar mais uma máquina, em cada *cluster*, a servir de *Grid Appliance*. Por fim esta arquitetura utiliza Globus Toolkit para gerir todos os recursos computacionais e tarefas [3].

O portal UCLA, como dito anteriormente, fornece um meio de acesso aos recursos comum, através de uma interface *web*. Com isso este é capaz de oferecer todo um conjunto de serviços, e esses são:

- Recursos: Cada utilizador tem a capacidade de ver o estado do *cluster*, se está ativo ou não, a sua carga, número de nós, número de tarefas a correr no momento e as que estão em fila de espera, etc. Assim os utilizadores têm um *feedback* sobre o estado dos *clusters* e podem interagir de acordo com a informação que obtêm [56].

- Gestor de dados: Este serviço oferece ao utilizador a capacidade de manipular ficheiros nos *clusters*: criar, apagar, modificar, mudar permissões, copiar etc. O gestor de dados oferece também visualização de diferentes dados *online* (sem os ter de descarregar para a sua máquina local), transferir dados entre diferentes máquinas ou mesmo carregar/descarregar ficheiros de/para a nossa máquina local [56].

- Serviços de tarefas: Este serviço dá a capacidade aos utilizadores de submeterem tarefas para os *clusters*. A submissão de tarefas pode ser dividida em duas categorias: *Generic Jobs* e *User Applications*. *Generic Jobs* dá a possibilidade ao utilizador de criar a sua própria aplicação. Sendo assim este terá que definir um conjunto de atributos necessários à definição da aplicação. Esses atributos podem ser, por exemplo, o caminho para o programa/executável, argumentos que sejam necessários, os ficheiros de entrada (*input*) e saída (*output*), entre outros. Já no caso das *User Applications* o utilizador pode "reciclar" tarefas que foram anteriormente submetidas por este. Sendo assim, o utilizador tem a possibilidade de re-submeter uma tarefa que já tenha sido submetida e de alterar qualquer parâmetro, se necessário, aumentando assim a sua produtividade. Por último o utilizador terá também a possibilidade de visualizar o estado das tarefas e descarregar o *OuputSandBox*, definido na aplicação, para a máquina local [56]. O atributo *OutputSandBox* é um conjunto de ficheiros definidos pelo utilizador a serem posteriormente descarregados para a sua máquina local ou algum servidor de armazenamento remoto. Normalmente, neste atributo são definidos os ficheiros que guardam os resultados da aplicação juntamente com um ficheiro de standard de erro, que guarda informação sobre algo que não tenha corrido nas melhores condições.

- Serviços Interativos: Este portal pode correr algumas aplicações de forma interativa tais como: sessão SSH em que o utilizador pode aceder a um *cluster*. Essa sessão é feita através de um *applet* Java. Escrever/compilar código C/C++/Fortran como um projeto *makefile*. Por último, permite correr aplicações gráficas interativas, bastando ao utilizador desbloquear as janelas *popup* do navegador *web* [56].

3.4.1.1 Vantagens

O portal *grid* UCLA oferece ao utilizador todo um conjunto de ferramentas permitindo estas interagir com os recursos existentes. Este tem a capacidade de submeter tarefas, verificar os diferentes estados das tarefas submetidas, transferir ficheiros, criar aplicações interativas etc. O portal fornece acesso seguro aos diferentes *clusters* com um único *username* e *password* (*Single Sign-On*) [53]. Outra vantagem é o portal ter sido criado completamente com *software open-source* tal como, Globus Toolkit [3], Tomcat, Java, Gridsphere e Mysql [54]. Uma outra vantagem é que sempre que um novo utilizador deseja utilizar o portal, poderá pedir um certificado à autoridade certificadora da própria universidade. A autoridade certificadora irá colocar o seu certificado (chave pública e privada) diretamente no servidor *MyProxy*, o que faz com que o utilizador nunca tenha de gerir o seu próprio certificado [79]. O portal UCLA oferece todo um conjunto de serviços bastante completo e que satisfaz as necessidades de praticamente todos os utilizadores.

3.4.1.2 Desvantagens

Apesar de oferecer todo um conjunto de serviços, este não tem a capacidade de atribuir certificados aos utilizadores de forma automática, pois estes têm de se deslocar a uma

autoridade certificadora por forma a provar a sua identidade. Em relação aos certificados, um outro problema é que sempre que o utilizador deseja aceder aos recursos fora do Campus, estes necessitam de pedir a um administrador que coloque um certificado válido, para os recursos a utilizar[79].

Uma outra desvantagem é também o desenho da página. Por exemplo, no serviço de submissão, o utilizador só deverá ter acesso direto aos atributos essenciais (básicos) para submeter uma tarefa. Muitas vezes os utilizadores não querem saber para onde vai parar a tarefa, qual o balanceamento dos *clusters*, prioridades etc. Eles somente querem submeter a sua tarefa de forma fácil e rápida e obter o resultado final. Como se pode ver na Figura 3.5, o utilizador comum não quererá saber com certeza a função dos atributos *Requirements* ou *Rank*, pois este só quererá submeter tarefas definindo o executável e os ficheiros de entrada/saída e pouco mais. Então para colmatar esse problema, os atributos referidos acima deverão aparecer sob a forma de uma extensão, ou seja, no caso do o utilizador possuir alguns conhecimentos de JDL, este poderá selecionar opções avançadas e definir novos atributos. Como podemos ver também a interface é um pouco confusa e não utiliza a tecnologia *portlet*.

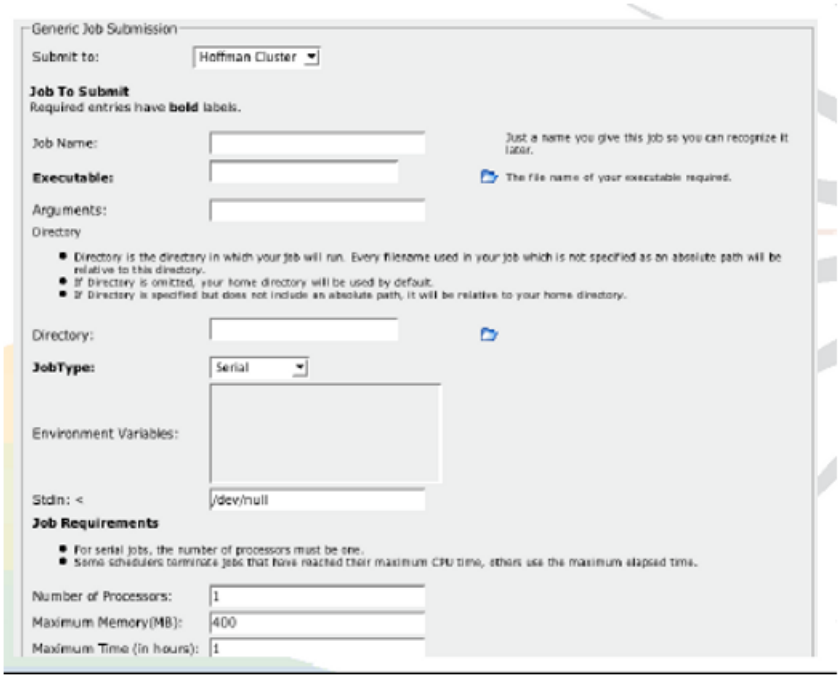

Figura 3.5: Submissão de uma tarefa genérica no Portal UCLA [53].

3.4.2 GENIUS

Tal como o portal anterior, o portal GENIUS permite que utilizadores devidamente registados possam submeter tarefas e oferece acesso a todo um conjunto de serviços.

Para os utilizadores que habitualmente utilizam o *middleware* LCG2-EGEE, estes estão habituados a ter todo um grande conjunto de ferramentas (linhas de comandos), para que possam interagir com a infraestrutura das mais variadas formas e de forma flexível. Como é de esperar, a curva de aprendizagem para a utilização destes serviços é grande, o que faz com que os projetistas da infraestrutura tenham de arranjar novas soluções para incentivar os utilizadores a usarem as infraestruturas de *grid*. Então a solução encontrada para esse problema foi a criação de um portal, em que os utilizadores possam facilmente (de forma gráfica) submeter tarefas, inspecionar os seus dados, descarregar/carregar ficheiros de/para a infraestrutura [67].

A arquitetura do portal GENIUS pode ser dividida em um modelo de três camadas, de acordo com a Figura 3.6:

- A camada cliente: Esta camada é onde os utilizadores fazem a interação através de um navegador *web*, sendo assim, o utilizador tem a liberdade de utilizar a infraestrutura em qualquer local desde que se autentique devidamente [68].
- Camada Servidor: Esta camada é a responsável por receber os pedidos vindos dos clientes, processá-los e encaminha-los devidamente para a infraestrutura. Esta serve também como *gateway* de interação entre o cliente e toda a infraestrutura existente. Esta camada servidor está equipada com um servidor *web* Apache, responsável por disponibilizar o conteúdo *web* e pela autenticação. Uma outra ferramenta é o Java/XML EnginFrame, dá ao utilizador a possibilidade de gerir todos os ficheiros da *User Interface* remotamente [67].
- Camada de recursos: Esta camada é a infraestrutura grid em si. Esta compreende todo um conjunto de *Computing Elements*, *Storage Elements*, *File Catalogs*, etc [67].

Para garantir a segurança de todas as transações a comunicação é feita usando SSL "Secure Socket Layer" via HTTPS. O utilizador para poder utilizar todas as potencialidades do portal terá de se autenticar duas vezes. A primeira autenticação serve para o utilizador poder aceder às funcionalidades do sistema operativo utilizado no portal e a outra serve para o que o servidor possa criar o certificado *proxy* a aceder a todas as funcionalidades da infraestrutura. O tempo de duração do certificado, por defeito, é de 12 horas, mas aqui o utilizador pode diminuir esse tempo aumentando assim a segurança [67].

Como é de esperar o portal oferece todo um conjunto de funcionalidades que vão ser referidas seguidamente. Então eis algumas dessa funcionalidades para dar a perceber o que é possível fazer com o portal GENIUS.

- Serviço de ficheiros: Este serviço dá a possibilidade ao utilizador de gerir todos os ficheiros na *User Interface*. Com este serviço o utilizador é capaz de criar, apagar, modificar, remover, listar ficheiros, além disso este poderá criar/apagar diretórios e carregar/descarregar ficheiros [77].
- Serviço de segurança: Este serviço é responsável pela criação de um certificado *proxy* para que o utilizador possa utilizar todos os recursos *grid*. Além disso, este serviço

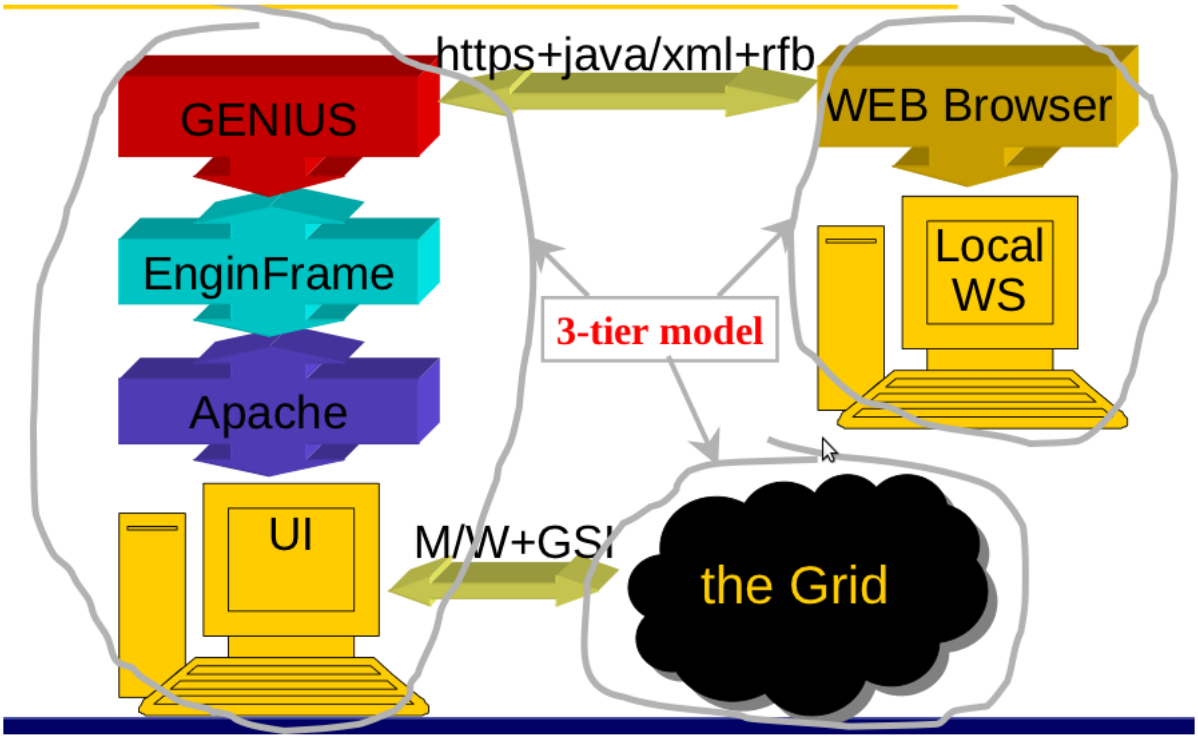

Figura 3.6: Arquitetura do Portal GENIUS [68].

permite ao utilizador verificar o seu certificado *proxy* temporário, dando informação como a validade, caminho onde se encontra localizado, inspecionar o seu certificado pessoal e de remover o certificado *proxy* criado [78].

- Serviço de Informação: Este serviço permite aos utilizadores obter informação geral sobre todos os recursos existentes numa organização virtual (VO), inclusive verificar informação da VO. A informação que poderá ser mostrada é, por exemplo, o número de CPUs, número de tarefas a correr ou que estão à espera num determinado CE, espaço livre de um SE etc [78].

- Serviço de monitorização: Permite aos utilizadores do portal monitorizar os recursos *grid* mais importantes. Com isto o utilizador é capaz de fazer uma análise superficial do estado da infraestrutura *grid*. Com essa análise o utilizador poderá calcular medidas como o comportamento, a performance e eventualmente detetar algumas falhas [78].

- Serviço VO: Este serviço é dividido em dois grandes serviços que são, gestor de tarefas e gestor de dados. O gestor de tarefas permite ao utilizador submeter tarefas e verificar o seu estado numa fila. Nessa mesma fila este pode encontrar outro tipo de informação como o identificador da tarefa, o CE onde está a correr etc. Este gestor oferece a possibilidade aos utilizadores visualizarem os seus dados diretamente no navegador *web*. O gestor de dados é o serviço responsável por toda a interação, do utilizador, com os ficheiros guardados no SE's. Este é capaz de submeter ficheiros para os SE's, criar réplicas e descarregar ficheiros para a sua própria máquina [77].

3.4.2.1 Vantagens

O portal GENIUS tem a capacidade de fornecer a todos os utilizadores um meio de utilização da infraestrutura de forma gráfica o que faz com que a curva de aprendizagem por parte destes seja menor. Sendo assim, praticamente qualquer utilizador está apto para submeter aplicações para a infraestrutura *grid*. Este, como vimos anteriormente, oferece todo um conjunto de serviços, em que o acesso é feito de forma confiável e segura.

Uma outra vantagem deste portal é a capacidade de mostrar os resultados de imediato para os utilizadores, ou seja, estes podem verificar todos os seus resultados sem ter a necessidade de descarregá-los para a sua máquina local. Para que isso seja possível o portal GENIUS integra uma ferramenta chamada TightVNC, responsável por mostrar graficamente para o utilizador resultados que estejam guardados na *User Interface*. A grande vantagem disto é a possibilidade de o utilizador manipular todos os seus dados remotamente sem ter que os descarregar para a sua máquina local, visto que estes dados normalmente são de grandes dimensões [67].

3.4.2.2 Desvantagens

Este portal tem duas grandes desvantagens, que são a necessidade de o utilizador ter de se autenticar sempre duas vezes e a forma como são submetidas as tarefas. Na caso da autenticação, para o utilizador é um pouco custoso estar repetidamente a introduzir as credenciais de acesso ao portal e para a criação do certificado *proxy* sempre que sai do portal. O certificado, quando é criado por defeito, tem um tempo de duração de 12 horas. Nesse período de tempo, o utilizador não deveria ter a necessidade de colocar as credenciais para a criação do novo certificado *proxy*, visto que o acesso está protegido com credenciais. No caso de submissão de tarefas, o ideal era existir só uma página em que o utilizador pudesse criar e submeter a sua tarefa e não ter que aceder repetidamente a diferentes páginas para criar e submeter uma tarefa. Como podemos ver na figura 3.7, apesar da interface ser simples, o utilizador tem que mudar várias vezes de página para definir os argumentos da tarefa, bem como os ficheiros de entrada, os requisitos, o *rank* etc. Um utilizador normal, não quer passar por ter de criar um diretório, um ficheiro JDL, introduzir um caminho para os ficheiros de entrada na UI etc. O utilizador, simplesmente, quer uma página simples onde possa definir a sua tarefa, carregar os seus ficheiros de entrada e carregar no botão submeter. Sendo assim o servidor deverá ficar responsável pela criação de todo o ambiente necessário à tarefa. Mais detalhes e uso avançado deveriam estar escondidos neste nível.

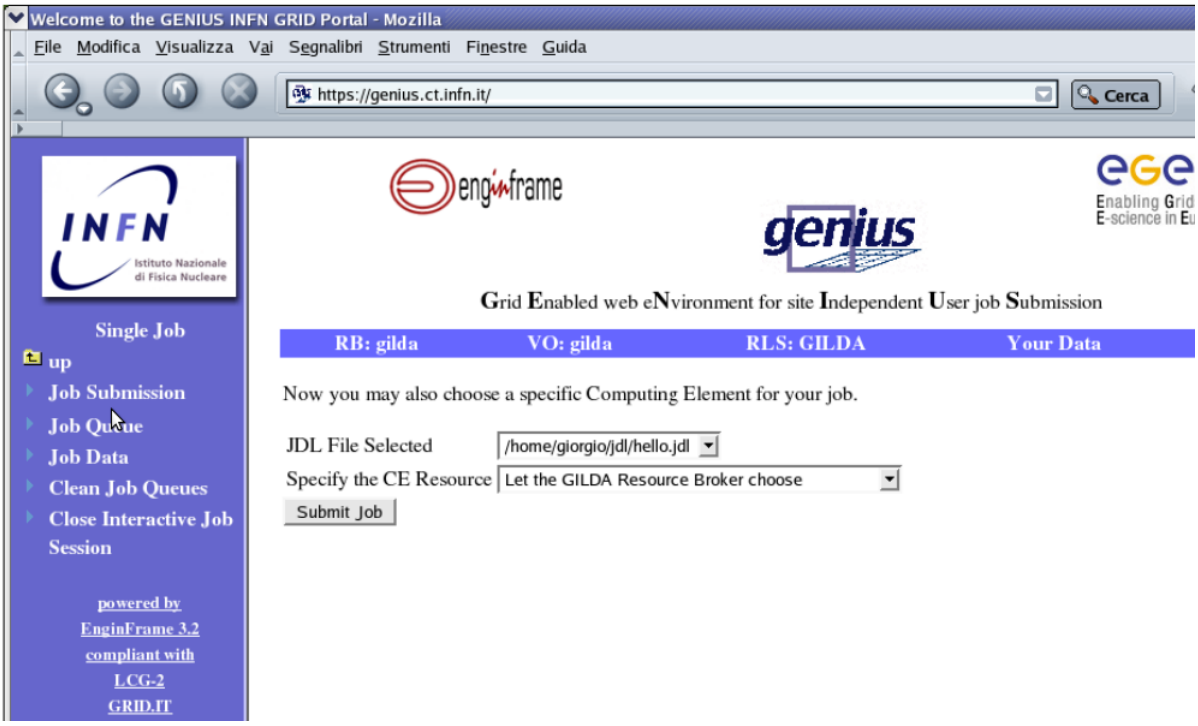

Figura 3.7: Interface do Portal GENIUS [78].

3.4.3 P-GRADE

O portal *grid* P-GRADE é um portal que permite aos utilizadores criar, monitorizar, executar tarefas numa infraestrutura *grid* usando uma interface *web*. Este portal foi desenvolvido pelo laboratório de sistemas paralelos e distribuídos de MTA-SZTAKI, Hungria [82]. O grande objetivo deste portal é esconder detalhes de acesso *grid* de baixo nível ao utilizador, oferecendo-lhe uma interface *web* para que este possa executar aplicações paralelas ou distribuídas em diferentes organizações de maneira fácil e rápida. A criação de tarefas é geralmente baseada em *workflows* em que os utilizadores definem as tarefas recorrendo a ferramentas gráficas de desenho [62]. Os editores de *workflows* mais utilizados são Taverna [86] e Triana [95]. Estes editores consistem numa interface gráfica que permite ao utilizador criar grupos de tarefas facilmente e submetê-los para a infraestrutura grid sem grande esforço [73]. O editor de *workflows* P-GRADE suporta tarefas Direct Acyclic Graph (DAG), em que os nós representam diferentes tarefas a ser submetidas. Os arcos do grafo representam os ficheiros necessários para a execução das tarefas, Figura 3.8. De notar que a tarefa só começa a executar no fim de todas as dependências estarem resolvidas. P-GRADE é um portal que oferece uma interface genérica para criação e execução de *workflows*, isto é, suporta vários tipos de *middleware* como gLite [19], Globus [75], NorduGrid [38] e gestor de *clusters* tais como PBS e LSF.

A arquitetura do portal P-GRADE divide-se em três grandes camadas, Figura 3.9. A primeira camada é composta pela interface com o utilizador, a segunda, pelo sistemas de ficheiros do servidor e a terceira pelas infraestruturas *grid* suportadas. O portal disponibiliza todo um conjunto de serviços através de *portlets*. Esses serviços são geridos

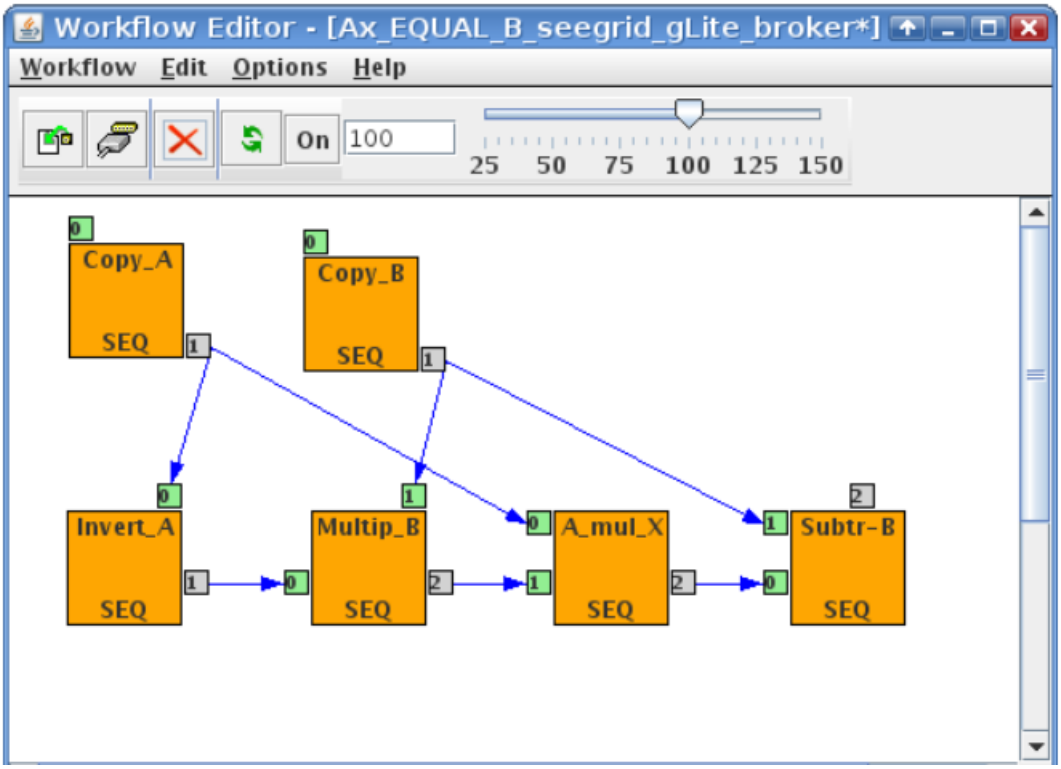

Figura 3.8: Criação de um *Workflow* no Portal P-GRADE [73].

através de alguns programas como Apache Tomcat e GridSphere. Apache Tomcat é o servidor HTTP/HTTPS responsável pela disponibilidade do portal na Internet, já GridSphere é o gestor de *portlets* que é responsável pela criação deste seguindo o standard JSR-168. Para a interação com o utilizador o portal disponibiliza de serviços [73]:

- Gestor de *Workflows*: Possibilidade de criar/gerir *workflows* de tarefas.
- Certificados: Responsável pelos certificados *proxy* e por carregá-los/descarregá-los.
- Sistema de Informação: Fornece a informação relevante sobre o estado dos recursos.
- Gestor de Ficheiros: Capacidade de gerir ficheiros no catálogo LFC.
- Repositórios de *Workflows*: Fornece a toda a comunidade uma ferramenta para que este possa partilhar todo o seu trabalho.

Quando se cria um *workflow* a linguagem gerada é comum, ou seja, é genérica, então a camada servidor é responsável pela conversão da linguagem genérica para as diferentes linguagens utilizadas nos variados sistemas. O portal P-GRADE gera uma linguagem de *workflow* chamada Condor DAGMan, um *script* deste tipo contêm toda a descrição do *workflow* criado, que posteriormente irá ser convertido para os diferentes sistemas *grid* como EGEE e ARC ou diretamente para a linguagem utilizada pelos gestores de recursos PBS e LSF. Para que tudo isto possa ser executado, o servidor corre um conjunto de

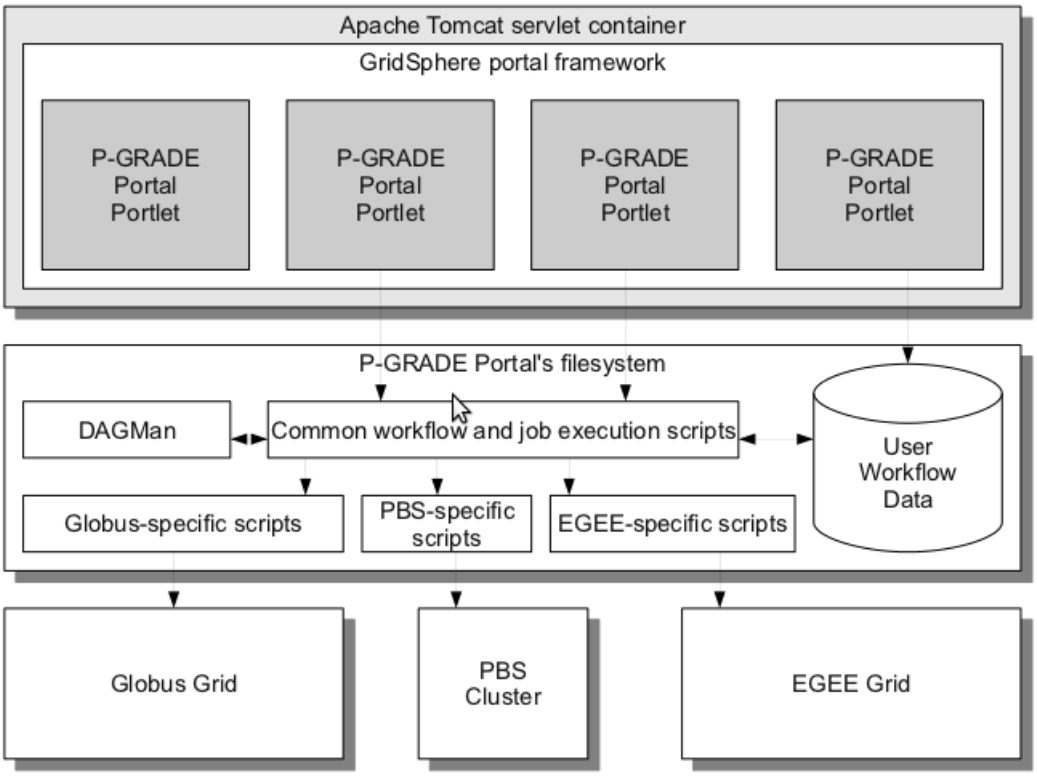

Figura 3.9: Arquitetura do Portal P-GRADE [73].

scripts conforme os pedidos dos utilizadores. De notar que a gestão de tarefas não é feita recorrendo a API's mas sim a *scripts shell* que vão correr diretamente no servidor [73]. Estes *scripts* são divididos em quatro grandes grupos.

- *Workflow Management Scripts*: *scripts* responsáveis pela gestão de todas as tarefas. Por exemplo, submeter, verificar o estado, cancelar tarefas.

- *Job Pre and Post Scripts:* Condor DAGMan utiliza um conjunto de *pre e post scripts* para gerir as tarefas dos utilizadores. Os *pre scritps* são responsáveis por juntar e processar os ficheiros de entrada pertencentes às tarefas, sendo assim também são responsável por preparar a tarefa para execução. Os post *scripts* são chamados quando as tarefas finalizam a sua execução e normalmente, processam os ficheiros de saída.

- *Grid-specific Job Managements Scripts*: Estes *scripts* são os responsáveis por adaptar o ficheiro de descrição genérico criado pelo portal para cada sistema *grid*. Por exemplo, converter o ficheiro para JDL para que possa ser executado num sistema grid que utilize o *middleware gLite*.

- *Grid-specific Job Wrapper Scripts*: Estes *wrapper scripts* são os *scripts* gerados automaticamente pelo portal, em que são submetidos para o CE para executar tarefas que a infraestrutura poderá não conseguir fazer. Por exemplo copiar ficheiros

de entrada remotos para o diretório de trabalho, carregar ficheiros para servidores remotos quando uma tarefa termina, etc. Estes *scripts* são essenciais para correr *legacy applications* já que normalmente são programas antigos que precisam de toda uma configuração de ambiente diferente. A estratégia é fazer com que estes *scripts wrapper* adaptem todo o ambiente sem a intervenção do administrador.

3.4.3.1 Vantagens

Uma das grandes vantagens deste portal é a capacidade do utilizador criar tarefas graficamente. Isso é possível recorrendo a um editor de *workflows* em que o utilizador define um conjunto de tarefas, que podem estar interligadas (dependências). Sendo assim o utilizador só tem que criar pequenos blocos quadrangulares, e para cada bloco definir o que quer executar. De notar que isto pode ser feito somente com o rato. No fim de submeter a tarefa este pode ir visualizando o seu estado, podendo mostrar qual a tarefa está a ser executada no momento.

Uma outra vantagem é o utilizador poder submeter o seu certificado pessoal para o servidor *MyProxy*. Como este portal utiliza várias infraestruturas *grid*, logo contem várias organizações virtuais (VO), então sempre que o utilizador precise de utilizar um VO diferente da que utiliza habitualmente, este poderá submeter uma tarefa para uma VO diferente. De notar que para utilizar uma VO diferente este deverá ter um certificado que permite utilizar as diferentes VO's ou mandar criar um novo certificado para a VO a ser utilizada. É também de referenciar que uma única infraestrutura *grid* contém, normalmente, várias organizações virtuais.

3.4.3.2 Desvantagens

A grande desvantagem deste portal é o facto de ter sido descontinuado. Sendo assim não irão lançar mais nenhuma versão de *software* para este portal. No caso de existirem *bugs*, estes não poderão ser corrigidos.

Uma outra desvantagem é a gestão de certificados não ser automática. Para o utilizador é bastante aborrecido ter que se deslocar a uma autoridade certificadora pedir por um novo certificado sempre que necessite utilizar uma organização virtual diferente. O ideal é o portal ter um serviço que através de uma autenticação federada instala o certificado pessoal num servidor *MyProxy* de imediato, sendo assim a infraestrutura estará pronta a ser utilizada logo de seguida. Sendo assim este não terá que esperar, pelo menos um dia, pela resposta e o certificado da autoridade certificadora.

3.4.4 GSG (GISELA Science Gateway)

O portal GSG é uma interface que fornece todo um conjunto de ferramentas, dados e aplicações a toda a comunidade pertencente. O grande objetivo deste, tal como os outros, é de oferecer uma interface simples e flexível aos seus utilizadores. Este tenta suprimir

as necessidades exigidas pelos utilizadores de maneira fácil, fazendo com que a curva de aprendizagem de utilização seja bastante reduzido. A infraestrutura utilizada é chamada de *e-infrastructure* em que consiste num grande conjunto de centro de dados interligados por NRENs "National Research and Education Networks" [66].

Este portal utiliza tecnologias recentes na sua construção, destacando-se o Liferay [89] e Shibboleth [50]. Com Liferay o programador pode facilmente criar *portlets* para a gestão e criação do portal. Esta tecnologia fornece todo um conjunto de ferramentas de autenticação e autorização, através da tecnologia SSO "Single Sign-On". O objetivo de SSO é de oferecer ao utilizador a capacidade de se autenticar uma única vez para aceder aos diferentes recursos e serviços. A tecnologia SSO que o portal GSG utiliza é o Shibboleth[22]. Shibboleth está divido em dois serviços principais IdP "Identity Provider" e SP "Service Provider". IdP é a componente responsável por fornecer as credenciais dos utilizadores. Cada organização gere o seu próprio IdP, que posteriormente poderá receber pedidos fora da sua organização. Os vários IdPs existentes formam um grande grupo e esse grupo é denominado de federação. O SP é o recurso da Internet a que o utilizador pode aceder, ou seja, o utilizador através de SP vai pedir permissões para utilizar um certo serviço/recurso mesmo que esteja fora do seu domínio administrativo [69].

Neste portal existe uma particularidade que o distingue dos demais, que é os utilizadores não têm a necessidade de submeter nenhum certificado pessoal para poder utilizar a infraestrutura *grid*. Os utilizadores somente com autenticação federada têm acesso aos recursos *grid*, facilitando assim o uso do portal, principalmente para utilizadores que não tenham experiência em utilizar *grid* e que não conhecem detalhes sobre segurança. Como dito anteriormente a autenticação feita por parte do utilizador é federada, então qualquer investigador/engenheiro pertencente a uma instituição, que forneça recursos computacionais ao projeto GISELA, poderá ter acesso ao portal bastando introduzir o *username* e *password* pessoal da organização e selecionar o país e a instituição a que pertencem [66].

O modelo de utilização é um pouco diferente dos restantes. Normalmente os utilizadores podem criar a suas próprias aplicações e submetê-las. No portal GSG o paradigma é um pouco diferente em que os utilizadores têm acesso a um grande repositório de aplicações, as quais estes podem selecionar para execução. Este portal fornece um vasto conjunto de aplicações para as mais variadas áreas de estudo e para os diferentes tipos de utilizadores. Sendo assim, o utilizador tem à sua disposição todo um conjunto de aplicações em que estes somente têm de introduzir os dados (p.e. ficheiros de entrada), colocar essa aplicação a correr e esperar pelo resultado final. Os utilizadores mais avançados podem configurar as aplicações modificando alguns parâmetros para que esta possa ser mais refinada e ir de encontro as suas necessidades. Este poderá também criar um ambiente virtual onde poderá preservar todos os dados gerados pelas suas aplicações. Esses dados poderão ser posteriormente partilhados entre os mais diversos utilizadores [66].

A estratégia adotada pelo GSG, como dito anteriormente, segue um caminho um pouco diferente dos portais enunciados anteriormente. Este portal tenta criar toda uma comunidade de investigação em que o seu ponto mais forte é a capacidade de partilha de projetos, de resultados em que todos podem utilizar aumentando assim o seu processo

produtivo. O espírito que este portal tenta incutir nos investigadores e o de colaboração, partilha, fortalecendo assim toda a comunidade científica que utiliza a infraestrutura *e-Infrastructure* para criação e exploração da ciência [66].

Um caso de uso muito utilizado neste portal é o diagnóstico da doença de Alzheimer e esquizofrenia. Normalmente um médico não possui conhecimento de programação, sendo assim o portal é capaz de fornecer uma aplicação em que este só tem que submeter uma imagem do cérebro do paciente em questão e deixar que a infraestrutura faça a comparação dessa imagem com centenas de imagens de pacientes saudáveis, permitindo assim dar um diagnóstico automático. Como é de esperar, este processo facilita imenso o trabalho de qualquer médico dando resultados muito mais rápidos e fiáveis [69].

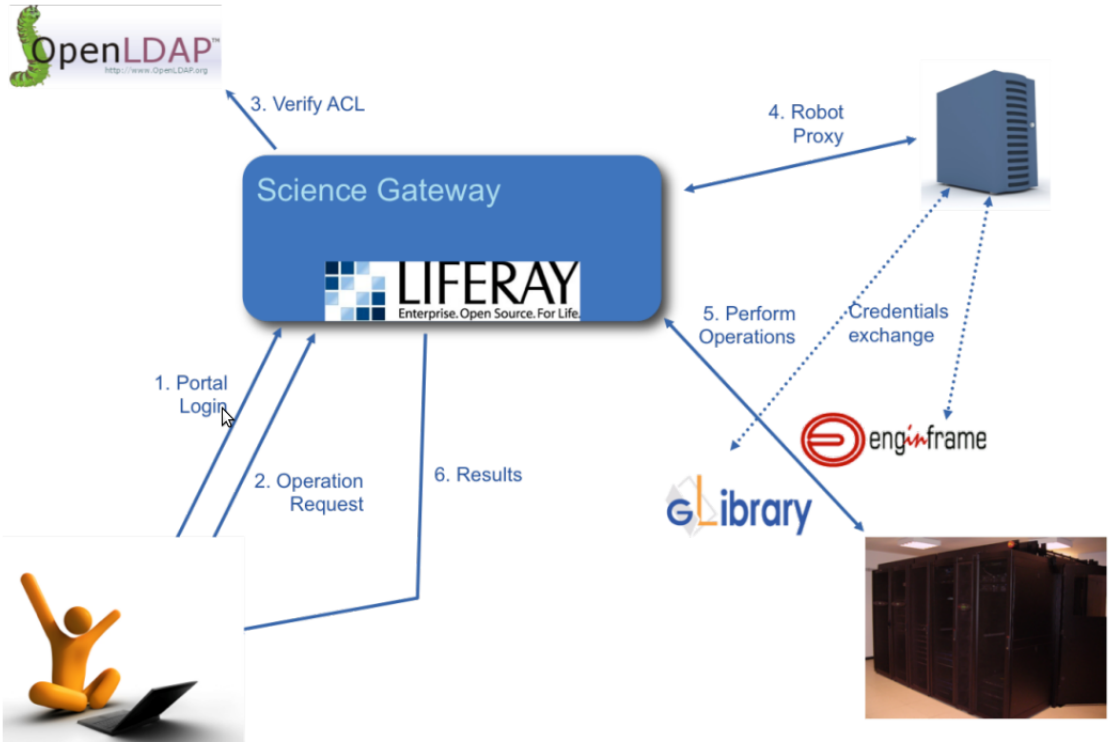

Figura 3.10: Arquitetura do Portal GSG [69].

Para compreender o uso do portal GSG, a Figura 3.10 mostra as interações, com os recursos *grid* principais. Quando um utilizador acede ao portal a primeira ação, por norma, a realizar é a autenticação (ação 1 – *Portal Login*). O portal GSG integra duas tecnologias que permitem a autorização e autenticação, sendo estas servidor LDAP (Lightweight Directory Access Protocol) [63] e Shibboleth [50] respetivamente. Sendo assim, quando o utilizador deseja se autenticar, o portal GSG irá redirecioná-lo para um processo de autenticação federada, onde este terá de escolher o IdP apropriado (nome da instituição pertencente). No fim da autenticação num IdP o portal irá aceder a um conjunto de dados localizados num servidor LDAP. Esses dados são a autorização do utilizador em questão para utilizar os recursos *grid*. De notar que se o utilizador não estiver registado nesse servidor o acesso é lhe negado. A autorização consiste em saber quais os membros do grupo, no qual o utilizador está registado. A partir da informação

recebida, o portal irá criar um conjunto de permissões automáticas para o utilizador em questão. O servidor LDAP é responsável por gerir grupos de utilizadores e de acordo com o grupo pertencente, as autorizações são atribuídas [69].

Como mencionado anteriormente, os utilizadores não necessitam de instalar certificados pessoais. A criação de certificados *proxy* temporários é feita através de certificados *robot* [72]. Os certificados *robot* são certificados especiais que permitem a criação de certificados *proxy* temporários. Sendo assim os utilizadores não necessitam de certificados pessoais para executar aplicações na infraestrutura. Quando um utilizador deseja executar uma operação na infraestrutura (ação 2 – *Operation Request*), o portal GSG irá verificar as autorizações obtidas (ação 3 – *verify ACL (Access Control List)*) e criar um certificado *proxy* temporário. A criação desse certificado é feita através do acesso a um servidor de *proxies* especializado (ação 4 – *Robot proxie*). De notar que para o mesmo serviço diferentes utilizadores poderão ter certificados *proxy* diferentes, vistos que estes podem ter privilégios diferentes. Por exemplo, um utilizador normal não tem as mesmas permissões que um administrador [69].

Os certificados *proxy*, criados através de certificados *robot*, não permitem identificar o utilizador, pois estes são iguais se diferentes utilizadores tiverem os mesmos privilégios. Então para os administradores poderem monitorizar toda atividade de determinado utilizador, os serviços registam todas as transações realizadas por determinado utilizador. Toda essa informação é guardada num sistema *User Tracking*, que combinada com informação existente no serviço LB podem efetuar associações entre utilizadores e operações de forma não repudiável [69].

Todas as restantes operações (ação 5 – *Perform Operation* e ação 6 – *Result*), são ações de execução de uma aplicação especifica e o seu respetivo resultado. Após a autenticação e autorização no portal o utilizador está apto a realizar operações com a infraestrutura grid subjacente [69].

Na Figura 3.10, no servidor *proxy* especializado estão representadas duas ferramentas gLibrary [18] e EnginFrame [12]. Glibrary é uma ferramenta criada pelo INFN, para criar e organizar ativos digitais (digital assets). Um ativo digital é conteúdo de determinado ficheiro que foi formatado em fonte binária que inclui direitos de uso [96]. Sendo assim, esta fornece de uma forma fácil e segura a gestão de ativos digitais, permitindo assim aos utilizadores gerir os seus ficheiros, como, por exemplo, fazer pesquisas avançadas, obter réplicas de ficheiros de um *Storage Elemnent* (SE) para a sua máquina pessoal e carregar ficheiros para os SE's [18]. EnginFrame é uma ferramenta escrita em Java que facilita o desenvolvimento de portais. Sendo assim esta oferece o meio seguro de acesso a todos os recursos *grid*, permitindo assim aos utilizadores submeter e controlar as suas aplicações *grid*, licenças, dados de forma fácil e segura escondendo a heterogeneidade e complexidade de todos os recursos existentes [69].

3.4.4.1 Vantagens

A grande vantagem deste portal é a capacidade de os utilizadores terem acesso aos recursos *grid* utilizando somente uma única autenticação federada. Assim os utilizadores são libertados da necessidade de ter um certificado pessoal.

Outra vantagem é a simplicidade de acesso pelos utilizadores sem experiência. O portal GSG fornece um conjunto de aplicações que estes podem utilizar de forma simples sem terem a necessidade de programar uma única linha de código. Este, então fornece todo um conjunto de aplicações especializadas a serem utilizadas, por exemplo em áreas como a medicina, biologia, física, bio-informática entre muitas outras. De notar que toda a informação apresentada é mostrada recorrendo à tecnologia *portlet*.

Para finalizar esta secção é importante enunciar a capacidade de partilha e de colaboração que este portal consegue oferecer aos seus utilizadores, aumentando assim a rentabilidade do trabalho contribuindo para o progresso da ciência mais rápido e sustentado.

3.4.4.2 Desvantagens

O portal GSG falha num dos principais objetivos do portal Grid UP, porque não permite aos utilizadores submeterem as suas próprias tarefas. Por mais aplicações que um portal possa ter, haverá com certeza a falta de outras aplicações. Se isso acontecer um utilizador não poderá criar a sua aplicação e submetê-la, o que retira alguma flexibilidade ao portal. Num portal normal um médico poderá também executar a aplicação de diagnóstico da doença de Alzheimer, desde que este tenha a aplicação na sua posse, bastando definila como o executável a correr com o seu respetivo *input*. Se optar por esta forma, o utilizador irá receber os mesmos resultados que se utilizarem GSG. Quanto às imagens de comparação estas poderão também já estar dentro da infraestrutura *grid*, visto que uma organização virtual permite partilhar todos os ficheiros.

O portal GSG é o ideal para os utilizadores inexperientes que normalmente utilizam quase sempre a mesma ferramenta para trabalhar. Já para utilizadores mais avançados este poderá ser um entrave, visto que estão limitando às aplicações existentes não podendo assim criar as suas próprias aplicações.

3.5 Conclusão

Como podemos ver existem diferentes portais para *grid*, mas nenhum satisfaz, completamente, os objetivos propostos. Além destes portais existem outros portais *grid*, mas os que foram enunciados são os mais relevantes e os que apresentam características próximas às desejadas. O estudo prévio destes portais serve também para complementar o portal Grid UP. Como é esperado, estes portais oferecem tecnologias inovadoras, que devem ser transpostas para o portal da Universidade do Porto. Essas tecnologias poderiam ser, a capacidade do utilizador criar tarefas baseadas num *workflow*, capacidade do utilizador ver

os seus resultados online sem os ter que descarregar para a sua máquina local etc. Neste capítulo, além dos vários portais existentes, vimos também a primeira e segunda geração de portais *grid*. Esta descrição é importante para que os programadores se apercebam das limitações que os primeiros portais tiveram e a forma como essas foram ultrapassadas. Como podemos ver, a tecnologia introduzida na segunda geração, *portlets*, é a base de criação dos portais *grid* e não só. Com portlets o programador concentra-se no conteúdo do portal deixando aspetos mais técnicos para o portal, tal como aspetos gráficos. Por fim a tecnologia *portlet* oferece todo um conjunto de ferramentas gráficas para que o utilizador possa personalizar a sua página de *portlets*, bem como toda a apresentação do *portlet*.

Capítulo 4

Portal Grid UP

4.1 Introdução

A EGI (European Grid Initiative) compreende vários países e instituições, entre eles Portugal e a Universidade do Porto. A Universidade do Porto, em particular, contribui com aproximadamente cem recursos computacionais para a Iniciativa Nacional de *grid* (Ingrid) e para a EGI. Estes recursos estão disponíveis e são utilizados por várias aplicações externas à Universidade do Porto, inclusive internacionalmente, mas são muito pouco utilizados e visíveis. Muitos investigadores na comunidade académica da Universidade do Porto poderiam beneficiar-se destes recursos se estes estivessem visíveis, por exemplo publicados em alguma página *web* e se o seu acesso fosse facilitado. Visto isto, desenvolvemos, portanto, um portal para a comunidade da Universidade do Porto que possui algumas das características dos portais estudados no Capítulo 3, com algumas vantagens específicas para a comunidade local.

O portal Grid UP é um portal que fornece aos utilizadores, pertencentes à Universidade do Porto, um meio de acesso aos recursos computacionais *grid* existentes sem grande esforço. A grande motivação deste serviço é melhorar a visibilidade da infraestrutura *grid* existente no Campus. Sendo assim, o grande objetivo deste portal é incentivar o uso de toda a infraestrutura por parte de utilizadores menos experientes (e mesmos os experientes), por forma a estes utilizarem recursos que estão disponíveis na sua própria instituição. Como é sabido existe uma certa complexidade no acesso à infraestrutura *grid* através da *User Interface*, então o portal Grid UP compromete-se a resolver esses problemas oferecendo ao utilizador final uma interface simples e clara. O portal disponibiliza um conjunto de serviços que funciona de forma transparente para o utilizador, ocultando complexidade técnica inerente por forma a incentivar o seu uso por parte daqueles que raramente/nunca utilizaram uma infraestrutura *grid*.

As tecnologias *web* têm emergido fortemente ao longo do tempo, fazendo com que a Internet seja mais rápida, fiável e robusta. Como hoje em dia existem conexões fantásticas à Internet (300Mbs, 1Gbs, linhas dedicadas, etc), seria um desperdício não a utilizar. A Internet permite a todas as pessoas visualizarem conteúdos, acederem a serviços, jogar

jogos *online*, entre outras muitas coisas, mas a grande vantagem é de podermos aceder a tudo isso independentemente do local que nos encontramos. Posto isto, o portal *grid* da Universidade do Porto não poderia deixar de utilizar este tipo de tecnologia, visto que é uma forma de disponibilização de serviços que muitas vezes precisam de ser acedidos a qualquer hora em qualquer local. Sendo assim, as tecnologias *web* têm inúmeras vantagens em relação às interfaces gráficas comuns que são só disponibilizadas no *desktop* do utilizador. Uma das grandes vantagens é permitir aos utilizador aceder a qualquer recurso, independentemente da localização geográfica que se encontra. Para o portal esta é uma caraterística insubstituível, visto que o portal terá de estar disponível 24 horas por dia e acessível em qualquer lugar. Outra vantagem é a capacidade de partilha. Nos dias correntes a maior parte dos projetos de investigação é realizado através de colaboração conjunta, o que torna uma mais valia poder partilhar informação num repositório comum. As desvantagens desta tecnologia poderá ter a ver com a segurança ou com a inatividade do serviço. As questões de segurança são ultrapassáveis desde que o utilizador seja cuidadoso na forma como acede aos serviços *web* e como define as suas palavras chaves no acesso. A inatividade do serviço pode ser um problema se o país ou instituição não oferecer conexões à Internet estáveis e rápidas, mas felizmente estamos numa instituição em que isso não será um problema. Visto isto, o portal da Universidade do Porto foi construído com tecnologias *web* que serão descritas nas secções seguintes.

Neste capítulo falamos sobre os requisitos do portal grid UP, sobre as diferentes tecnologias existentes para a sua construção, as suas funcionalidade e também o porquê da opção recaída sobre as tecnologias utilizadas. Este capítulo está organizado da seguinte maneira. Na secção 4.2 discutimos os requisitos do portal como os tipos de utilizadores e de aplicações. A secção 4.3 enumera as tecnologias existentes e o porquê da escolha das tecnologias utilizadas. A secção 4.4 descreve todas as funcionalidades existentes no portal Grid UP e por fim a secção 4.5 conclui este capítulo.

4.2 Levantamento de Requisitos para a Construção do Portal Grid UP

Com base na experiência da EGI e nas aplicações desenvolvidas em infraestruturas de *grid*, fizemos um levantamento das necessidades de aplicações e recursos para a construção do portal. Enumeramos os requisitos da seguinte forma:

- O portal deverá ser capaz de melhorar a visibilidade da infraestrutura, como forma de incentivo à sua utilização.
- Este terá que fornecer apoio ao utilizador, ou seja, qualquer conteúdo apresentado deverá ser devidamente explicado por forma a melhorar a usabilidade da interface.
- Divulgar o estado da infraestrutura *grid*. Este deverá ser capaz de fornecer *feedback* explicativo sobre algo que possa não ter corrido nas melhores condições. Por exemplo, o utilizador deverá saber quando a infraestrutura está *offline*, sendo assim

esta é incapaz de executar tarefas. De notar que a infraestrutura grid muito esporadicamente é que não está disponível.

- Possibilitar a visualização das tarefas, ou seja, o utilizador deverá ter a possibilidade de verificar o estado das tarefas submetidas.
- Capacidade de submissão de tarefas de forma fácil. O utilizador menos experiente deverá ser capaz de submeter uma aplicação definindo unicamente o nome do executável, carregar os ficheiros que interessam e pouco mais.
- Deverá existir mecanismos de armazenamento de informação de forma estruturada, por forma a que cada utilizador tenha o seu ambiente de trabalho organizado e separado dos restantes.
- Deverá existir um método da acesso à infraestrutura *grid*. Através dele, o utilizador será capaz de submeter as suas tarefas bem como geri-las.
- Diminuir a burocracia existente na utilização das infraestruturas *grid*, principalmente a nível de gestão e pedido dos certificados pessoais.
- Permitir só autenticação federada (através das credenciais do sistema sigarra), para que só utilizadores da Universidade do Porto, numa fase inicial, possam usufruir da infraestrutura *grid*.
- Deverá oferecer uma interface de fácil utilização, intuitiva e sem grandes detalhes técnicos gráficos para poder oferecer uma interface simples e limpa.

4.2.1 Tipos de utilizadores

A população alvo deste portal, são todos os investigadores/engenheiros pertencentes a toda uma comunidade académica. Essas áreas podem ser, matemática, física, biologia, bioquímica, medicina, ciência de computadores, astronomia entre outras. Num grupo de diferentes áreas de estudo, os utilizadores que normalmente estão mais acostumados a utilizar a infraestrutura *grid* são os de física, matemática e ciências de computadores, pois estes a utilizam mais frequentemente e estão mais familiarizados a trabalhar com sistemas operativos da família do UNIX. Uma das grandes barreiras na utilização da infraestrutura *grid* é o facto dos utilizadores não terem conhecimento sobre navegar num ambiente *shell*. Um outro obstáculo existente, está diretamente relacionado com o anterior, é a gestão do certificado pessoal que é demasiado burocrático na sua forma atual. Por essas razões existe a necessidade de criação de um portal para que todos os utilizadores possam ter acesso aos recursos computacionais da sua própria instituição de forma simples e à distância de um simples *click* do rato.

No inicio desta secção foram enumerados alguns serviços básicos, os quais o portal Grid UP se deve reger. Sendo assim os utilizadores deverão ser capazes de se familiarizar rapidamente com a interface e esta deverá ser intuitiva para que a curva de aprendizagem de utilização seja bastante reduzida. Se o processo de utilização da infraestrutura for

muito complicado e burocrático os utilizadores certamente irão procurar outras soluções. O incentivo à utilização de toda a infraestrutura computacional disponibilizada pela Universidade do Porto é importante por três razões. A primeira é que se existirem inúmeros utilizadores a usar os recursos disponibilizados, estes poderão ser expandidos por forma a socorrer as necessidades internas e também assim tornar a infraestrutura mais robusta e poderosa. A segunda razão é o facto de que se a instituição tem recursos computacionais e estes não são aproveitados, isto leva a que haja desperdício de energia elétrica e também a que alguns projetos de investigação tenham que adquirir novas máquinas ou alugar computação aumentando os custos. A terceira razão seria atrair a utilização dos recursos da Universidade do Porto por investigadores externos. Desta forma, a instituição poderia ter receita adicional alugando tais recursos a qualquer investigador/engenheiro externo.

4.2.2 Tipos de aplicações

Nesta secção falamos sobre a taxonomia dos sistemas *grid*. As taxonomias são divididas em três categorias – *grid* computacional, *grid* de dados, *grid* de serviços. *Grids* computacionais disponibilizam grande capacidade de processamento às aplicações. Sendo assim, estes sistemas podem ser sub-divididos em *distributed supercomputing* e *high throughput computing*. Uma grid *distributed supercomputing*, normalmente, executa aplicações em paralelo em diversas máquinas, isto para reduzir o tempo de execução de determinada tarefa. Já uma grid *high throughput computing* aumenta a taxa de execução de um conjunto de tarefas, ou seja, maximiza a taxa de execução de tarefas por unidade de tempo [83].

Grid de dados é para sistemas que forneçam uma infraestrutura de repositórios de dados, que estão distribuídos geograficamente. *Grids* computacionais também precisam de fornecer dados. A grande diferença entre elas é que *grid* de dados é uma infraestrutura que fornece aplicações especializadas de gestão de armazenamento e acesso a dados. Um exemplo dessas aplicações serem especializadas é o facto de elas, por exemplo, usarem mineração de dados para correlacionar informação de diferentes fontes de dados [83].

Grid de serviços é para sistemas que forneçam serviços que são disponibilizados por um conjunto de máquinas. Esta categoria é sub-dividida em três sub-categorias - sistemas *on-demand*, colaborativos e de multimédia. Sistemas *on-demand* permitem dinamicamente agregar diferentes recursos para disponibilizar um serviço. Por exemplo, nesta categoria um investigador pode alocar mais recursos computacionais dinamicamente para tornar mais realista simulações meteorológicas. Sistemas colaborativos permitem interligar utilizadores e aplicações em grupos de trabalhos. Sendo assim, os utilizadores têm interação em tempo real com as aplicações. Sistemas de multimédia fornecem uma infraestrutura em tempo real de conteúdos multimédia [83].

Os tipos de sistemas *grid* que os investigadores da Universidade do Porto usualmente utilizam são: *grid* computacionais e de dados. Sendo assim, o portal deverá suportar a utilização desses tipos de sistemas para fazer face a todas as necessidades. Atualmente, o portal Grid UP, somente disponibiliza *grid* computacional, mas a extensão para grid de dados é relativamente trivial. Os utilizadores podem submeter tarefas do tipo *Normal*,

paralelas (*MPICH*) ou paramétricas. Tarefas paramétricas são tarefas que podem ser submetidas ao mesmo tempo apenas com variação dos parâmetros.

4.3 Tecnologias Disponíveis para a Construção do Portal

De momento, encontra-se disponível no mercado uma vasta coleção de *software* capaz de fazer cumprir os requisitos necessários para a construção de um portal *grid*. No mercado existe quer *software* proprietário quer *software* livre que satisfaz todas as necessidades enunciadas. Optamos por usar *software* livre, visto que este não acarreta qualquer custo e o programador poderá estudar o seu código fonte, alterá-lo e distribuí-lo, normalmente sob as mesmas condições de licença [16].

O desenvolvimento do portal passa por várias escolhas por exemplo, de sistema operativo, gestão de base de dados, plataformas de desenvolvimento de portais *web*, *middleware* de grid, linguagens de programação para o cliente e para o serviço, dentre outras. Abaixo enumeramos o *software* mais popular utilizado para cada uma destas classes.

- Sistema operativo. Windows Server, Linux.
- Sistema de gestão de base de dados. Mysql, Oracle, Microsoft SQL Server, PostgreSQL.
- Plataformas de desenvolvimento de portais web. Liferay, Gridsphere.
- Meios de acesso à infraestrutura *grid* (*middleware* gLite). jLite, User Interface API's.
- Linguagens de programação (servidor). Java, PHP, Perl, Python.
- Linguagens de programação web. HTML, JSP, Javascript.
- Bibliotecas de desenvolvimento web. JQuery, Vaadin, Yahoo YUI.
- Outras linguagens. Expect, Script Shell, CSS.

Como mencionado na introdução deste capítulo, a nossa escolha para a criação do portal Grid UP foi a disponibilização do mesmo através de serviços HTTP, fazendo com que este esteja sempre disponível em qualquer lugar. Todo o desenvolvimento do portal Grid UP recorreu a uma plataforma de desenvolvimento de portais, Liferay [89], juntamente com a plataforma Eclipse [10], facilitando o desenvolvimento de aplicações *web*. Todo o portal Grid UP é escrito na linguagem de programação Java [29], pois esta permite grande interoperabilidade com Liferay e Eclipse. Todas as escolhas efetuadas serão discutidas de seguida. No que se segue discutiremos sobre as tecnologias disponíveis e aquelas escolhidas para a construção do portal *grid*.

4.3.1 Sistemas Operativos

O sistema operativo adotado para a construção do portal da Universidade do Porto foi Linux. A escolha recaiu sob o facto do sistema Linux ser o mais usado para o suporte às páginas *web*, cerca de 63,9 por cento de todas páginas *web* existentes [55]. Uma outra razão para a escolha deste sistema operativo é que este é distribuído sobre a GNU GPL (General Public License), ou seja, é um *software* livre que não acarreta qualquer custo. Então como Linux é um *software* livre os programadores têm a liberdade total para estuda-lo, modificá-lo, introduzir melhorias, conforme as suas necessidades, e liberdade para distribuir um programa alterado (melhorada) desde que seja sob as mesmas condições de licença [21]. Já com um sistema operativo Windows isso já não é possível, visto este ser proprietário e o programador não tem liberdade de fazer as modificações que deseja. Para perceber melhor as diferenças entre Linux e Windows, de seguida são enumeradas alguns pontos essenciais que fazem o sistema operativo Linux mais apetecível para ambientes servidor:

- Estabilidade: Um sistema Linux fornece ao utilizador grande estabilidade, podendo frequentemente correr durante meses e anos sem ser reiniciado. Já um sistema Windows não consegue oferecer tal estabilidade [49]. Segundo um estudo disponível em [49] cada servidor Linux está em baixo (*downtime*) durante trinta a sessenta minutos por ano, enquanto cada servidor Windows, em média, está em baixo durante nove horas. Este ponto reforça ainda mais a escolha sobre o sistema Linux para o portal Grid UP.
- Segurança: Uma das grandes razões para um sistema Linux ser mais seguro que Windows é o facto de existir uma grande comunidade que suporta o próprio sistema. Sempre que existe algum problema toda a comunidade junta esforços, por forma a tentar resolvê-lo. Um sistema Linux também é vulnerável a ataques, mas a grande flexibilidade de configuração do sistema e a comunidade existente permite fazer com que essas vulnerabilidades sejam mais rapidamente ultrapassadas [27].
- Preço: Como já mencionado nesta secção Linux não custa nada, bem como muito do software utilizado. Em Linux encontra-se uma grande variedade de ferramentas sem nenhum custo e com qualidade. Já em sistemas Windows é mais difícil encontrar muitas ferramentas sem nenhum custo.

Apesar de todas as vantagens enunciadas, existem também algumas desvantagens que muitas vezes fazem com que os programadores/administradores optem por sistemas Windows. A grande desvantagem é a capacidade de uso. Num sistema Windows, os administradores têm interfaces gráficas que permitem gerir toda uma coleção de ferramentas, enquanto num sistema Linux muitas das ferramentas, principalmente de administração, só estão acessíveis a partir da linha de comandos. Sendo assim, a curva de aprendizagem na utilização de um sistema Linux é bastante mais elevado que no Windows. No entanto, tal complexidade permite disponibilizar um sistema mais robusto e fiável (Linux). Outra desvantagem é a compatibilidade de *hardware*. Muitas vezes o *hardware* não é compatível com sistemas Linux o que obriga o utilizador a usar sistemas Windows. Estas são desvantagens que não afetam diretamente o desenvolvimento do portal.

4.3.2 Sistema de Gestão de Base de Dados

O sistema de gestão de base de dados escolhido foi MySQL. MySQL é o sistema de gestão de base de dados relacionais *open source*, mais usado no mundo, desenvolvido e distribuído pela Oracle [47]. A Oracle é uma multinacional Norte-Americana especializada em sistemas de hardware e produtos de software empresarial [41]. MySQL oferece acesso multi utilizador a inúmeras bases de dados. Base de dados relacionais são coleções de dados organizados em tabelas, em que o seu acesso é simples [36]. O código fonte do projeto MySQL está disponível sob os termos de GNU GPL "General Public Licence", podendo ser distribuído e modificado sob a mesma licença GPL [37]. Este sistema de gestão de base de dados é multi-plataforma, o que é vantajoso, visto que todo o portal Grid UP pode facilmente ser migrado para outro sistema operativo como, por exemplo, Windows.

Os sistemas de base de dados Oracle e Microsoft SQL server não foram escolhidos, visto serem *software* pago. As tecnologias a utilizar no portal devem ser *open source* como mencionado anteriormente, apesar do sistema de base de dados Oracle ser bastante eficiente e utilizado pelas grandes companhias. O investimento não se justificaria visto que este sistema de base de dados é utilizado em aplicações que necessitam de grande escalabilidade e performance [43]. Apesar do sistema da base de dados PostgreSQL ser também *open source*, este não foi escolhido visto ser um pouco mais lento que MySQL no acesso [71]. Para o propósito do portal Grid UP, MySQL é melhor visto que o objetivo é guardar somente alguma informação não complexa. Se, por exemplo, desejássemos guardar informação espacial teríamos de usar PostgreSQL, visto que esta tecnologia suporta informação espacial, oferecendo funções, por exemplo para cálculo de distâncias, rotas, entre outras.

4.3.3 Plataformas de Desenvolvimento de Portais *Web*

A escolha da plataforma a utilizar não é algo muito relevante, visto que ambas, Liferay e Gridsphere, suportam portlets JSR 186 e JSR 286. Os *portlets* desenvolvidos podem ser migrados entre as duas plataformas, visto que os portlets foram criados para interoperar em diferentes plataformas. A plataforma escolhida foi Liferay e a grande motivação desta escolha foi a sua excelente documentação e fácil instalação. Além disto, o desenvolvimento da plataforma Gridsphere foi descontinuado.

Liferay Portal é uma tecnologia *open source* escrita em Java que contém um grande número de ferramentas, que ajudam os programadores a implementar a sua interface *web* no menor tempo possível [89]. O portal Liferay é distribuído sob a licença LGPL (GNU Lesser General Public Licence) e proprietário. Existem duas distribuições diferentes deste produto. A primeira é *Liferay Portal Community Edition* - edição livre com as últimas atualizações e suportada pela comunidade [33]. A segunda é *Liferay Portal Enterprise Edition* - edição comercial que inclui atualizações e suporte completo, como por exemplo formação de uso da tecnologia.

A apresentação do conteúdo *web* pode ser feito através de *portlets* ou *OpenSocial Gadgets*.

Os *portlets* são aplicações *web* escritas em Java que correm numa porção da página *web*. Os *portlets* funcionam de forma independente de todo o portal e a comunicação entre cliente e servidor é feita através de *servlets*. Um *servlet* é uma classe Java usada para estender capacidades de servidores HTTP. Através deste o programador é capaz de gerir (receber e responder) pedidos HTTP [60]. Com *portlets*, o programador terá só de se concentrar no conteúdo da página *web* e na definição do servidor, deixando detalhes de organização e apresentação para o portal Liferay. *OpenSocial Gadgets* permite ao programador integrar aplicações *web* tradicionais com *portlets*, como por exemplo, integrar a página de um jornal desportivo. *OpenSocial Gadgets* é uma forma de criação de *portlets* recorrendo somente à localização dos recursos *web*. Estes recursos *web* poderão ser externos, por exemplo criar dois *portlets* que inclua notícias de dois jornais distintos na mesma página. Essa inclusão de páginas é feita recorrendo ao URL (Uniform Resource Locator) da mesma. Através destas tecnologias, os criadores de páginas *web* têm à sua disposição ferramentas para criação de páginas de forma simples e rápida. Estes são capazes, também, de integrar serviços remotos já previamente criados na sua interface [92].

O portal Liferay oferece ao programador a capacidade de saltar inúmeros passos técnicos, o que faz com que a criação de páginas seja um processo mais rápido e simples. Com Liferay o utilizador pode criar portlets, personalizá-los, gerir comunidades/organizações, criar grupos de utilizadores, atribuir diferentes permissões a diferentes conteúdos de uma mesma página, isto tudo graficamente. Além disso, as seguintes características tornaram o portal Liferay muito popular [91]:

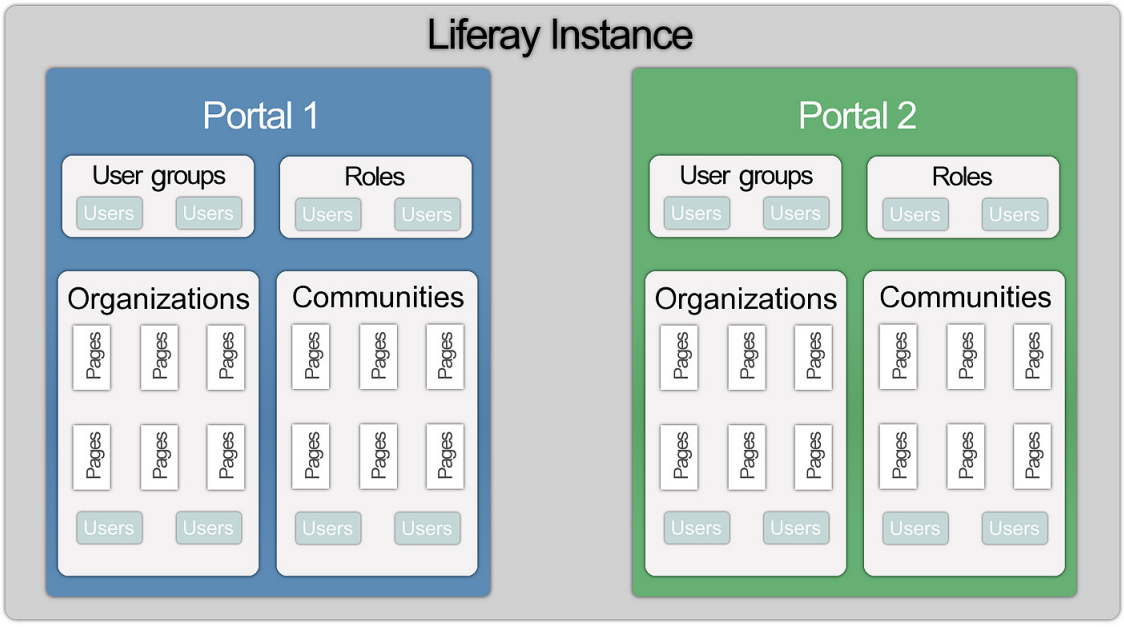

Figura 4.1: Coleções de utilizadores em Liferay [89].

- Caracterização de utilizadores: Os portais têm utilizadores, e com esta tecnologia estes poderão ser caracterizados em várias coleções, Figura 4.1. Essas coleções permitem gerir os diferentes grupos de utilizadores e adaptar o portal aos diferentes tipos de utilizadores. Eis os tipos de coleções [89]:

  - *Role*: Agrupam os utilizadores pela sua função, por exemplo agrupar utilizadores que tenham as mesmas permissões.
  - *Organization*: Organiza os utilizadores pela posição na hierarquia. Sendo assim consegue atribuir as mesmas permissões ao grupo de utilizadores pertencentes à mesma organização.
  - *Community*: Utilizadores com interesses comuns. Por exemplo, podem ter permissões de acesso a determinada página se pertencerem a uma comunidade específica.
  - *User Group*: Permite criar uma coleção de utilizadores com o mesmo propósito. Por exemplo, pode existir um grupo de utilizadores para criação de *blogs* ou redes sociais, sendo assim estes necessitaram de permissão especiais para poder efetuar tais tarefas.

- Segurança: O portal Liferay utiliza tecnologias de encriptação comuns, como DES, MD5 e RSA. Este permite também SSO "Single Sign-On" para fazer autenticação federada em diferentes servidores de autenticação. Além disto, este oferece também toda a criação e gestão de contas de utilizadores, verificação de e-mail e gestão de sessões. De notar que as sessões expiram automaticamente após passado algum tempo de inatividade [91].

- Gestão de temas e personalização: O utilizador tem a liberdade de modificar os temas se assim o desejar. Inicialmente, o portal Liferay fornece alguns temas base para que os programadores possam utilizar. O administrador não necessita de reiniciar o servidor para modificar o tema, essa modificação é feita em tempo de execução da aplicação servidor. Para aplicações mais avançadas o portal oferece ao utilizador a capacidade de definir o seu próprio tema manualmente. Os programadores podem definir os seus próprios temas definindo o *layout* da página, a forma como é organizado, as cores, entre outras especificações técnicas de desenho [91]. Para cada *portlet* apresentado na página, o utilizador pode modicar o seu aspeto independentemente. Por exemplo, poderá modicar as cores, margens, tipo e tamanho do texto. Além disto o administrador poderá definir as permissões que deseja para cada *portlet*, por exemplo definir que só uma determinada organização pode visualizá-lo [89].

- Colaboração: Dentro do portal Liferay todos os utilizadores podem ter acesso a *blogs*, *messaging*, redes sociais. Com isto estes podem falar com os seus parceiros de trabalho como estivessem a utilizar o MSN e partilhar o seu trabalho entre os demais utilizadores [91].

- Compatibilidade: Este portal é compatível com os diferentes sistemas operativos (bastando ter a plataforma Java instalada), com os diferentes tipos de base de dados (bastando definir o caminho da classe Java, o URL, *username* e *password* de acesso), e com diferentes aplicações servidor bem como Apache Tomcat e Glassfish [91].

- Instanciar *portlets*: Os utilizadores podem adicionar os *portlets* que desejarem, desde que tenham permissões para tal. Os utilizadores têm uma barra horizontal no topo

da página, onde podem selecionar os *portlets* disponíveis e adicioná-los [89]. A Figura 4.2 mostra um exemplo de lista de portlets que podem ser adicionados.

- Painel de Controlo: Este é uma interface para administração do portal como, por exemplo, adicionar páginas, gerir o servidor de email, juntar utilizadores às diferentes comunidades. O painel de controlo está dividido em quatro secções, sendo estas [89]:
  - *User Administration*: Aqui os utilizadores podem gerir toda a sua informação pessoal, bem como modificar informação da sua conta e adicionar/gerir as suas páginas pessoais. Basicamente isto é uma área reservada a cada utilizador.
  - *Content Administration*: Esta divisão contém um conjunto de funções de gestão de conteúdo. Esse conteúdo pode ser, por exemplo, gerir páginas *web*, documentos, imagens, calendário, *bookmark*, etc.
  - *Portal Administration*: Secção destinada à configuração do portal, ou seja, modificar as configurações necessárias. Aqui os administradores podem gerir as organizações, comunidades, permissões e grupos de utilizadores.
  - *Server Administration*: Esta secção oferece ferramentas para gerir questões administrativas. Essa ferramentas são: modificar o servidor SMTP e POP de email bem como as credenciais de acesso e as respetivas portas, modificar o tamanho máximo de ficheiro que pode ser carregado, executar funções de limpeza à memória do servidor entre muitas outras coisas.

O portal Liferay contém muitas outras características, mas estas são as mais importantes porque realçam a capacidade de realizar inúmeras tarefas sem recorrer à programação de raiz. Para finalizar este ponto, é importante realçar que a disponibilização de conteúdo poderá ser feito nos demais navegadores *web* existentes, quer em computadores pessoais quer em *smartphones*, fazendo então a renderização automática para cada plataforma e resolução [92].

4.3.4 Meios de Acesso à Infraestrutura *Grid*

A opção sobre qual o meio de acesso às funcionalidades *grid* recaiu sobre a biblioteca Java jLite. O grande potencial desta, em relação à API gLite localizada na *User Interface*, é o facto de ser portátil, funcionar em qualquer sistema operativo e ser fácil de configurar, além de não exigir a existência de uma máquina extra para a interface (User Interface).

JLite é uma API (Application Programming Interface) Java , que permite os programadores acederem às funcionalidades da infraestrutura *grid* através da linguagem de programação Java. O *middleware* que esta biblioteca pode interagir chama-se gLite. O grande objetivo desta biblioteca é reduzir o tempo e esforço por parte dos administradores em adaptar uma plataforma que interaja com o *middleware* gLite. Além de disponibilizar uma biblioteca compatível com a linguagem de programação Java, esta fornece toda uma interface de linha de comandos onde os utilizadores podem aceder à infraestrutura *grid*

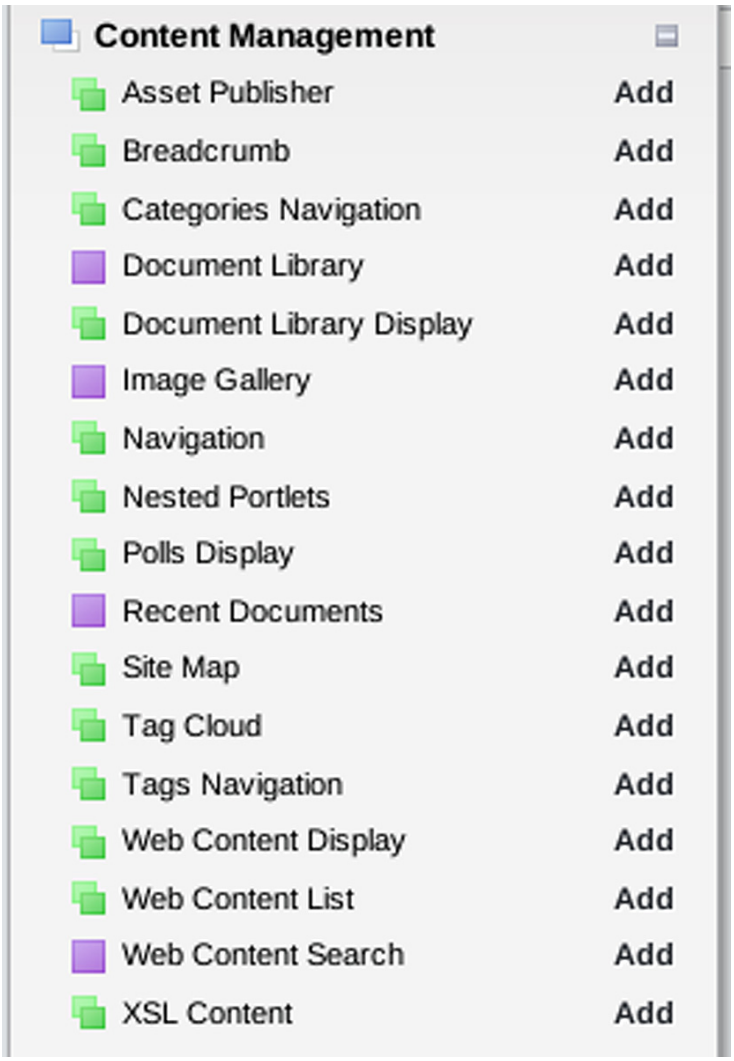

Figura 4.2: Adicionar *portlets* em Liferay [89].

(simula uma *User Interface*). A interface de linha de comandos está disponível quer para o sistema operativo Windows quer para Linux, disponibilizando scripts *.bat* e *.sh*, respetivamente para tal efeito [94].

Atualmente, existem diversas bibliotecas Java que interagem com o *middleware* gLite [90]. Essas bibliotecas não descartam a presença de um ambiente *User Interface* gLite. A implementação corrente desta biblioteca permite gerir todo o ciclo de vida de uma tarefa, simulando assim as funções de linha de comando da *User Interface*, e de submeter vários tipos de tarefas, bem como *Normal*, *Collection* e *Parametric*. Algumas funcionalidades que esta biblioteca é capaz de suportar são [94]:

- Criação e delegação da *proxy* VOMS (Virtual Organization Membership Service): O utilizador desde que tenha o seu certificado pessoal, é capaz de criar e delegar o seu certificado *proxy* para a infraestrutura. Sendo assim este está apto para executar qualquer função na infraestrutura.
- Transferir ficheiros de *input*: O utilizador só precisa definir o diretório onde se encontram localizados todos os ficheiros de entrada (executável, ficheiros JDL etc). Após a definição desse diretório a biblioteca automaticamente irá submetê-los para o local correto na infraestrutura.
- Submeter tarefas: Os utilizadores submetem as suas tarefas, desde que tenham o ficheiro JDL previamente definido.
- Monitorização de tarefas: No fim das tarefas submetidas, cada utilizador pode recorrer a esta funcionalidade para verificar o estado da sua tarefa. A identificação da tarefa é feita através de um *id* único que é gerado quando a mesma é submetida.
- Obtenção dos ficheiros de *output*: Assim que as tarefas terminem, os resultados podem ser transferidos para a máquina local. Aqui o utilizador tem a necessidade também de passar o identificador da tarefa.

A vantagem de utilizar esta biblioteca é ser multi plataforma, funcionando em diferentes sistemas operativos como Windows, Linux, Macintosh. Estes sistemas operativos só necessitam ter o JDK (Java Development Kit) instalado para que ela funcione, isto porque ela é totalmente desenvolvida em Java. Já as outras bibliotecas necessitam mesmo de uma *User Interface* para que tudo funcione corretamente. A API jLite esconde toda a complexidade subjacente da configuração de acesso ao *middleware* gLite. Esta é bastante fácil de instalar, porque inclui todas as dependências externas necessárias e funciona em qualquer computador com Internet, assim não necessita de nenhuma *User Interface* instalada. A sua instalação é simplesmente descomprimir o arquivo onde se encontra, configurar alguns ficheiros e está pronta a usar [94]. Para a configurar o administrador só necessita de configurar três ficheiros, sendo estes o ficheiro de definição VOMS, da VO (Virtual Organization) e WMS (Workload Management System). Basicamente aqui os administradores só têm que definir caminhos para os servidores, as portas onde estão à escuta, a organização virtual e a autoridade certificadora.

A desvantagem que poderá existir, é o facto dos utilizadores quererem fazer mais do que a biblioteca tem para oferecer, por exemplo gerir ficheiros nos SE (Storage Elements). Mas isto poderá ser facilmente resolvido, bastando expandir a biblioteca (já que o código é aberto) ou mudar toda a configuração para uma *User Interface* gLite e utilizar API gLite. De notar que a configuração jLite é portátil bastando copiar o diretório para outra máquina, definir o diretório do certificado pessoal e está pronto a utilizar.

4.3.5 Linguagens de Programação (servidor)

A linguagem de programação escolhida para acolher o servidor foi Java. A escolha recaiu sob o facto de a tecnologia Liferay ser feita em Java o que permite grande interoperabilidade, além desta suportar outras linguagens de programação como PHP, Ruby e Python [45]. Esta plataforma disponibiliza todo um conjunto de bibliotecas que permite ao programador gerir e criar páginas *web* de forma simples e eficaz. Para tornar mais forte a opção da escolha, a API jLite é escrita em Java, logo a linguagem para o servidor teria de ser obrigatoriamente Java, pois sem esta não poderia existir a interoperabilidade entre a infraestrutura *grid* e o portal *grid*.

Tanto o lado do servidor quanto o lado do cliente são cem por cento Java. Para o lado do cliente foi utilizada uma ferramenta que permite compilar Java para linguagens web como HTML, Javascript e CSS. Esta ferramenta é chamada Vaadin e vai ser descrita na secção a seguir. Java é uma linguagem bastante robusta, intuitiva e organizada. Esta linguagem disponibiliza todo um conjunto de bibliotecas e documentação bem estruturada e explicativa na Internet. Em relação à organização do conteúdo, Java permite gerar automaticamente documentação sobre o código de forma simples. Essa documentação é chamada de *Java Docs* que compreende todo conteúdo explicativo descrito pelo programador em cada função. *Java Docs* é um documento HTML, gerado automaticamente, para que os programadores possam verificar a função de cada método Java de forma simples. Normalmente o que é mostrado neste tipo de documentação é a função de cada método juntamente com os argumentos necessários, a explicação dos mesmos e também o tipo de valor que é retornado [28].

4.3.6 Bibliotecas de Desenvolvimento *Web*

Para desenvolvimento web poderíamos ter utilizado várias tecnologias diferentes, tais como HTML, JSP, Javascript. A escolha recaiu sobre uma ferramenta Java chamada Vaadin, pois esta permite ao programador desenvolver páginas *web*, recorrendo somente à linguagem Java. Esta ferramenta consegue gerar código web (HTML e Javascript) mais eficiente do que um programador comum poderia criar [26]. Além disso, uma das vantagens de utilizar esta ferramenta é a simplicidade de uso. O programador escreve código, cliente e servidor, todo junto como se tratasse de um ficheiro Java tradicional.

Vaadin é uma ferramenta Java de desenvolvimento de aplicações web para RIA (Rich Internet Application). Aplicações RIA são aplicações *web* com as mesmas características

do *software* que se encontra nos computadores pessoais. Sendo assim, RIA permite ter aplicações bastante interativas num navegador *web* e com a vantagem de poder ser acessível de qualquer ponto do mundo [48]. Para mostrar os dados no lado do cliente Vaadin utiliza a ferramenta GWT (Google Web Toolkit) [26]. GWT é uma ferramenta que permite otimizar aplicações *web* sem que o programador tenha noções de Javascript e comunicações AJAX. Sendo assim esta permite otimizar aplicações *web*, fazer *debug*, utilizar Java API e *widgets* para criação de páginas *web* [22]. Um *web widget* é uma pequena aplicação que pode ser instalada e executada dentro de uma página *web* por um utilizador por exemplo, como se fosse um *portlet*. É bastante comum encontrar *widgets* de informação de tempo, horas, informação sobre voos, eventos, entre outros mais [65]. AJAX (Asynchronous JavaScript and XML) é uma técnica, usada no lado do cliente, para comunicação assíncrona com o servidor. Sendo assim, com esta poderemos enviar e receber dados de forma assíncrona, fazendo com que a página *web* esteja sempre operacional mesmo quando está a haver troca de informação [14]. A tecnologia Vaadin veio revolucionar o desenvolvimento *web*, visto que basicamente estamos a gerar código Java completamente normal, como se tratasse de um programa comum [26]. O programador assim não terá a necessidade de conhecer as diferentes linguagens existentes para o desenvolvimento *web*, bastando saber Java.

A construção de páginas *web* em Vaadin é muito similar à construção de interfaces gráficas em Java, mas a única diferença é que este gera código *web* ao invés de *bytecode*. Bytecode é um conjunto de instruções que a máquina virtual Java executa. O programador tem à sua disposição um conjunto de componentes, tais como *Buttons*, *Tables*, *TextFields*, *Labels*, que pode utilizar na construção da sua interface *web*. Tal como acontece nas interfaces gráficas, em Java o programador pode usar eventos, *listeners*, *data binding* para comunicar entre as várias componentes e com o servidor. Todos esses eventos são geridos pela ferramenta Vaadin [57]. Um pequeno exemplo de utilização de Vaadin é mostrado na Figura 4.3. O exemplo, ilustrado na Figura 4.3, mostra o código Vaadin para criação de um *portlet* contendo uma tabela com três colunas, *Job Name*, *Job ID* e um botão *Submit*. As linhas 5-12 mostram a inicialização das componentes necessárias, que têm a seguinte função pela ordem respetiva: obter o *id* do utilizador, criar uma janela, criar um acesso à base de dados, criar uma tabela e inicializar o cabeçalho da tabela. A linha 15 mostra a função utilizada para obter todas as tarefas de determinado utilizador. De notar que a classe *MysqlAccess* já foi inicializada com o *id* do utilizador, logo esta vai retornar as tarefas do utilizador em questão. As linhas 16-23 mostram um ciclo, responsável por percorrer todo o *resultSet* retornado pela função anterior e preencher a tabela com esses dados. Entre as linhas 18-20 o programador tem a possibilidade de programar uma parte do servidor, mais precisamente tratar eventos vindos do botão *Submit*. Com este tipo de flexibilidade temos a possibilidade de programar cliente e servidor no mesmo ficheiro. Podemos considerar todo o código que não está entre as linhas 18-20 código cliente. A linha 22 mostra a tabela a ser preenchida com conteúdo da base de dados (*job_name* e *job_id*) juntamente com um botão criado na linha 17. Por último a linha 26 adiciona à janela a componente *queueTable* completamente inicializada e preenchida. A janela *mainWindow* é objeto da área *portlet*, ou seja, todo o conteúdo está contido nesse objeto.

Em relação ao aspeto gráfico, as várias componentes apresentam um *layout* e apresentação

```
1     <import libraries>
2     public class JobQueue {
3       public void init() {//portlet programming
4         //Initialization
5         int userID = (int) ((User) getUser()).getUserId();
6         Window mainWindow = new Window("Job_queue Application");
7         MysqlAccess mysqlDB = new MysqlAccess(userID);
8         Table queueTable = queueTable = new Table("Job Queue");
9         //init heads of "queuetable" (Job Name, Job ID, Submit)
10        queueTable.addContainerProperty("Job Name" ,String.class, null);
11        queueTable.addContainerProperty("Job ID" ,String.class, null);
12        queueTable.addContainerProperty("Submit" ,Button.class, null);
13        //get results from BD for specific user id
14
15        ResultSet resultSet = mysqlDB.getJobsList();
16  while(resultSet.next()) { //while exist tasks
17          Button submit = new Button("Submit"); //init "submit" button
18          submit.addListener(new Button.ClickListener() {
19            //do something. Event Handling. Server side
20          });
21          //add row to "queueTable" with results from BD and one button
22          queueTable.addItem(new Object[] {resultSet.getString("job_name"),
                resultSet.getString("job_id"), submit, jobName);
23          }
24    resultSet.close();
25          //add table to main window
26          mainWindow.addComponent(queueTable);
27        }
28      }
```

Figura 4.3: Exemplo Java com Vaadin

por defeito, mas todo o aspeto gráfico das componentes poderá ser modificado usando CSS (Cascading Style Sheets). Por defeito cada componente traz com ela classes CSS já pré-definidas para que todo o aspeto possa ser alterado usando tais classes. As classes aqui enunciadas são equivalentes às classes que se define numa *tag* HTML, que tem como objetivo representar a *tag* para poder ser acedida tanto por CSS como por Javascript. O programador tem também a possibilidade de definir o seu próprio Javascript recorrendo às bibliotecas YUI Yahoo! (The Yahoo! User Interface Library) e jQuery. Estas bibliotecas vêm por defeito instaladas, mas o programador tem a liberdade de instalar outras, se necessário [80].

A vantagem de utilizar esta ferramenta é a capacidade de gerar código Javascript (*client-side*) de forma eficiente e a rapidez com que uma página pode ser criada. Outra vantagem é a modularização de código, assim o programador é capaz de organizar todo o seu código de forma elegante, legível e clara, permitindo assim uma melhor reutilização/compreensão. Para finalizar, Vaadin pode ser integrado numa plataforma Eclipse ou Netbeans, para rápido desenvolvimento e melhor organização dos projetos criados [57].

4.3.7 Outras Linguagens (Expect)

Expect é uma linguagem que estende a linguagem tcl, que serve para automação de programas interativos e está disponível para os sistemas operativos Windows e Linux. Tcl é uma linguagem de *scripting* usada para os mais variados fins como por exemplo, administração, redes, *web*, aplicações *desktop*, interfaces gráficas, entre outros [52]. Expect veio trazer alguma flexibilidade ao programador que necessite de fazer chamadas a programas interativos. O tipo de programas que podem ser usados para tirar partido das potencialidades desta ferramenta são por exemplo *telnet*, *ftp*, *passwd*, *fsck*, *rlogin*, *tip*, entre outros [15].

No portal Grid UP esta ferramenta é utilizada para separar as chaves públicas e privadas dos certificados pessoais dos utilizadores da infraestrutura *grid*. Sendo assim, com esta, o portal é capaz de executar as funções anteriores automaticamente sem a intervenção do utilizador, permitindo assim automatizar o processo de submissão de certificados com um certo nível de segurança. Um pequeno exemplo de utilização da ferramenta Expect é mostrado na Figura 4.4:

O programa, representado na Figura 4.4, é utilizado para separar a chave pública de um certificado em formato *.p12*. Um ficheiro no formato *.p12* é um certificado que contém a chave pública e privada de um utilizador. Normalmente este é protegido por uma palavra chave, posteriormente necessária para separar as respetivas chaves privada e pública [46]. O comando *set timeout 30* (linha 5) faz com que o programa Expect espere pelo menos 30 segundos, se nada acontecer este termina imediatamente. Isto serve para que o programa não fique a correr indefinidamente. Os comandos *set certPath [lindex $argv 0], set outFile [lindex $argv 1] , set password [lindex $argv 2]* (linhas 7-9) atribuem valores das variáveis onde vão ficar guardado o caminho para o certificado, o ficheiro, neste caso, onde vai ficar guardada a chave pública e a palavra chave de exportação especificada no navegador. De notar que esses valores serão passados como argumento ao programa Expect. O

```
1   #!/bin/sh
2
3   package require Expect
4
5   set timeout 30
6
7   set certPath [lindex $argv 0]
8   set outFile  [lindex $argv 1]
9   set password [lindex $argv 2]
10
11  spawn openssl pkcs12 -clcerts -nokeys -in $certPath -out $outFile
12
13  expect {
14    "Enter Import Password:" {send $password\n}
15  }
16
17  close $spawn_id$
```

Figura 4.4: Caso de utilização da ferramenta Expect

comando *spawn* vai iniciar um processo para correr o programa *openssl*, que cria a chave pública. Recorrendo ao comando *spawn* (linha 11) é possível passar valores aos programas interativos – comunicação de processos. O comando *expect {...}* (linhas 13-15) vai esperar por uma string vinda do processo acima. Então quando o programa *openssl* pedir por uma *password*, este irá esperar pela *string, "Enter Import Password:"* e posteriormente irá enviar uma palavra chave para o certificado *send $password*. A variável *$password* (linha 9) é definida acima e o seu valor é passado por argumento de linha de comando. Por fim o comando *close $spawn_id* (linha 17) espera que o processo termine e mata o processo que foi lançado para correr o programa *openssl* [15] .

4.3.8 Outras Linguagens (CSS)

CSS (Cascading Style Sheets) é uma linguagem *style sheet* usada para descrever a apresentação de um documento escrito numa linguagem de marcação. Com esta linguagem o utilizador tem a capacidade de gerir toda a apresentação de uma página *web* como as cores, *layout*, posicionamento, etc [8]. Normalmente, o conteúdo HTML e CSS encontra-se separado por diversas razões. Com o conteúdo separado fica mais fácil para os programadores fazer a manutenção das páginas *web*, permite partilhar ficheiros *style sheet* com outras páginas e permite adequar as páginas a diferentes ambientes alterando só o necessário [23].

Com Vaadin todos os elementos criados vêm com uma apresentação por defeito, mas é possível para o programador alterar toda essa apresentação se assim o desejar. Para cada elemento é definido um conjunto de classes para que possam ser acedidas através dos ficheiros CSS. Então a partir dessas classes o programador pode descrever o elemento (só o aspeto gráfico) da forma como bem entender.

A vantagem de utilizar esta linguagem *style sheet* é que esta permite ao programador definir todo o aspeto e organização de uma página *web* e separar todo o código do restante. Uma outra vantagem é o facto de esta ser totalmente livre (não paga) e ter uma excelente documentação na Internet.

4.4 Funcionalidades

Como mencionado anteriormente, o portal Grid UP possui um conjunto de serviços básicos em funcionamento que serão descritos nas seguintes secções.

4.4.1 Autenticação

No momento, a autenticação possível é a autenticação tradicional onde o utilizador faz um pedido ao portal por credenciais, preenche um formulário e posteriormente irá receber um email de confirmação para poder utilizar o portal. Após a confirmação, cada utilizador tem a sua própria área pessoal onde pode modificar toda a informação pessoal sempre que quiser. Este tipo de autenticação não é a desejada, mas sim a autenticação federada através das credenciais da plataforma *sigarra* da Universidade do Porto, que irá ser posta em prática só quando este estiver em produção num servidor da Reitoria. Sendo assim, só os utilizadores pertencentes à instituição é que teriam permissões de acesso ao portal. A autenticação federada permite também, que os utilizadores não tenham que definir mais um novo *username* e *password*, pois normalmente as pessoas possuem dezenas de credenciais em locais diferentes.

Mesmo se o tipo de autenticação fosse a normal (se o portal estiver em produção), os utilizadores fora da instituição não poderiam aceder à infraestrutura, pois necessitariam também de um certificado pessoal o qual é preciso provar a identidade numa Autoridade Certificadora ou numa Autoridade de Registo como por exemplo, a Reitoria da Universidade do Porto.

4.4.2 Informação

A informação que está disponível ao utilizador, atualmente, é a informação sobre os recursos *grid* existentes na Universidade do Porto e uma breve descrição sobre a infraestrutura que a compõe. Na informação sobre os recursos é mostrada a localização dos *clusters* dentro do Campus e algumas caraterísticas técnicas. Já sobre a infraestrutura, são enumerados os elementos da arquitetura como por exemplo *Computing Element*, *Storage Element*, *Logging and Bookeeping*, *Workload Management System*, etc. Para cada elemento é feita uma breve descrição sobre a sua função na infraestrutura *grid*.

4.4.3 Gestão do Certificado Pessoal

Este serviço é essencial para que os utilizadores consigam ter acesso à infraestrutura *grid* e para que tenham as permissões necessárias para executar as suas tarefas e realizar ações. Para os utilizadores que tenham em sua posse um certificado válido, estes o poderão submeter através do portal sem nenhum problema. Para aqueles que ainda não o possuam, estes terão que se deslocar a uma autoridade certificadora ou autoridade de registo para provar a sua identidade. Só no fim da sua identidade provada e de um formulário preenchido é que estes poderão prosseguir com o pedido. Com o pedido feito, o utilizador terá de esperar 24 horas por um email vindo da autoridade certificadora. No fim do email recebido o utilizador irá clicar num *link* disponibilizado para que o certificado possa ser instalado no seu navegador *web*. Quando o certificado estiver instalado, no navegador, o utilizador terá que exportá-lo para o seu computador pessoal para que possa posteriormente submetê-lo para o portal Grid UP. Ao exportar o certificado, o navegador irá obrigar o utilizador a definir uma palavra chave para proteção do mesmo. Este processo é burocrático, mas atualmente temos que lidar com ele.

O serviço de gestão de certificados permite ao utilizador submeter os certificados pessoais de duas maneiras distintas, que são submissão de certificados no formato *.pkcs12* e no formato *.pem*. Por defeito os certificados exportados do navegador vêm no formato *.pkcs12*, sendo assim o utilizador poderá submeter este ficheiro. O portal irá separar a chave pública da privada automaticamente. O utilizador só necessita introduzir a chave de exportação e definir uma chave nova para proteger a chave privada oriunda do certificado. Se o utilizador optar por submeter o certificado no formato *.pem*, este terá de fazer a separação de chaves manualmente e posteriormente submeter a chave pública e privada. Esta última opção poderá ter alguma vantagem sobre a outra, porque assim o utilizador não tem a necessidade de definir a chave de proteção da sua chave privada, através do navegador o que lhe permite ter maior segurança. De notar que estas transações irão ser todas executadas através de SSL (Secure Socket Layer). Além disto, este serviço fornece toda a informação necessária sobre como pedir o certificado, o formulário a preencher para o pedido com a respetiva ajuda, o local onde se devem deslocar para o pedido, informação de como extrair o certificado do navegador *web* e informação de como separar as chaves públicas e privadas do certificado manualmente.

O serviço de gestão de certificados pessoais tem como objetivo simplificar a submissão de certificados e a sua gestão. Para muitos utilizadores é difícil submeter certificados via SSH, mais precisamente com o programa SCP (Secure CoPy), através do ambiente *shell* do sistema operativo Linux. Mas o ideal era ter uma forma de reduzir a burocracia no pedido do um certificado pessoal sem que o utilizador tenha de se deslocar a nenhuma autoridade certificadora.

4.4.4 Submissão de Tarefas

Como dito anteriormente, o portal Grid UP permite a todos os utilizadores, devidamente autenticados e com o certificado pessoal instalado, submeter tarefas para a infraestrutura

grid pertencente à Universidade do Porto. O objetivo deste serviço é o de facilitar o processo de submissão de tarefas por parte dos utilizadores menos experientes e aumentar a rentabilidade na criação de tarefas. Sendo assim todos eles têm à sua disposição uma interface, uma espécie de formulário, para preencher dando origem a uma tarefa que posteriormente será submetida.

O serviço de submissão está dividido em duas partes principais, uma de criação de tarefas (definição) e a outra para carregar os ficheiros necessários para a tarefa. Na primeira parte, o utilizador tem à sua disposição um formulário, pré-preenchido, para que possa definir a sua própria tarefa. O pré-preenchimento do formulário serve para poupar tempo ao utilizador na definição de atributos que muitas vezes são iguais como por exemplo, nome dos ficheiros de erro e saída e definir o *OutputSandBox*. Esses atributos são mostrados ao utilizador para que ele os possa modificar, se necessário, e também para servir de ajuda para compreensão do atributo em questão. Esta parte, ainda apresenta três botões com as seguintes funções:

- *Add Element*: Este botão serve para os utilizadores avançados, onde estes podem adicionar mais atributos e descrever melhor a sua tarefa. Com este, eles podem, por exemplo, especificar o CE de destino, definir prioridades para a sua tarefa, etc.
- *Show JDL File*: Este botão é utilizado simplesmente para que o utilizador possa ver o ficheiro JDL criado. Assim o utilizador poderá ter a noção da sintaxe da linguagem JDL e da sua estrutura.
- *Reset Window*: A função deste botão, tal como o nome indica, é fazer com que toda a página volte às configurações iniciais.

A segunda parte deste serviço serve para que os utilizadores possam carregar todos os ficheiros necessários à tarefa. Os ficheiros normalmente que serão submetidos são ficheiros de *input* para algum programa/executável e os próprios programas/executáveis. Esta parte tem outra funcionalidade que permite ao utilizador submeter um ficheiro JDL, fazendo então com que o formulário se preencha automaticamente. Isto poderá ser importante porque se o utilizador tiver em sua posse um ficheiro JDL, basta submetê-lo e está pronto a executar sem ter que preencher nada. Uma outra função deste serviço é de oferecer a capacidade ao utilizador de carregar *templates* de tarefas, agilizando o processo de criação de tarefas. Uma outra função que este serviço deveria ter, e atualmente não tem, é a possibilidade de criação de *templates*. É possível carregá-los, mas a sua criação tem de ser feita à mão em vez de ser automática. Imaginemos que um utilizador cria uma tarefa complexa, em que tem de utilizar muitos atributos e que estes têm alguma complexidade, seria interessante que o utilizador pudesse guardar a estrutura de toda a tarefa para que possa usar futuramente.

4.4.5 Fila de Tarefas

A fila de tarefas serve de registo de todas as tarefas submetidas ao longo do tempo. Atualmente só são suportadas tarefas do tipo *Normal*. Cada utilizador pode verificar

todas as tarefas submetidas e executar ações sobre elas. A fila de tarefas fornece ao utilizador informações sobre cada tarefa separadamente, que poderão ser importantes. As informações dadas são:

- *Job Name*: O *Job Name* permite identificar uma tarefa através de um nome legível, definido pelo utilizador. Quando é submetida uma tarefa é gerado um *id*, *string* aleatória, que identifica uma tarefa. Para um utilizador que tenha muitas tarefas na fila, torna-se difícil identificar a tarefa através do *id*, visto que é uma *string* aleatória extensa. Logo se um utilizador definir um nome para as tarefas fica mais fácil a identificação das mesmas.

- *Job id*: É um URL com uma *string* aleatória gerada automaticamente após a submissão de uma tarefa na infraestrutura *grid*. Esta *string* serve de identificação da tarefa na infraestrutura para que possa, por exemplo, verificar o seu estado de execução e obter o *output*. Com este *id* o utilizador pode saber toda a informação sobre a tarefa respetiva, em qualquer local. Inclusive com esse URL o utilizador pode aceder, através do navegador, à informação respetiva da tarefa desde que tenha o certificado pessoal instalado no navegador.

- *Submission Time*: Aqui, neste ponto, é registado a data e a hora de quando uma determinada tarefa foi submetida para a infraestrutura. A referência temporal é feita recorrendo ao relógio do próprio servidor.

- *Conclusion Time*: Aqui, neste ponto, são registados a data e a hora de uma determinada tarefa. Sempre que uma tarefa atinge o estado *DONE* é registada a hora dessa ação. Tanto este tempo como o anterior são calculados recorrendo à classe Java *Date*, que retorna o tempo atual do servidor. O tempo indicado não corresponde ao tempo real de execução da tarefa, este é mostrado para que os utilizadores tenham a noção do de execução da sua tarefa.

- *Status*: O *Status*, como o nome indica, serve para o utilizador saber o estado corrente da sua tarefa. O estados possíveis definidos no portal são, *READY_TO_SUBMIT*, *RUNNING_ON_GRID*, *ABORTED*, *CANCELED_BY_USER* e *DONE*. Os estados enunciados anteriormente são estados definidos para o portal e não estão relacionados com os estados de execução de uma tarefa. Existe um conjunto de estados que aparecem na fila de tarefas que não estão aqui representados, que são os estados de execução quando estas estão em execução na infraestrutura *grid*, ver Capitulo 2 secção 2.7 para ver os diferentes estados possíveis. Esses estados vão, então, substituir o estado *RUNNING_ON_GRID*. Estes grupos de estados criados servem para o portal moldar a interface conforme o estado apresentado e também para oferecer ao utilizador uma perceção mais clara sobre o que está a acontecer no momento.

No parágrafo anterior, foram enunciadas algumas informações que a fila de tarefas apresenta ao utilizador. Neste agora, são descritas as ações que cada utilizador é capaz de executar sobre cada tarefa. Eis então algumas ações disponíveis:

- *Submit*: Tal como o nome indica, o utilizador ao clicar neste botão está a dar ordem para que a tarefa referida possa ser submetida para a infraestrutura *grid*. Se o utilizador tiver um certificado *proxy* válido, o portal não irá pedir pela criação de um, caso contrário este irá pedir a palavra chave para que possa criar um certificado *proxy* válido. De notar que a validade de um certificado *proxy* é de 12 horas, ou seja, durante esse tempo o portal não irá pedir novamente por credenciais. Sempre que o utilizador clica neste botão é registado na informação o *id* da tarefa e a data e hora de submissão, *Submission Time*.

- *Update*: Este botão serve para atualizar o estado de uma determinada tarefa. Sempre que um utilizador clica neste botão está a fazer um pedido à infraestrutura, mais precisamente ao serviço *logging and bookkeeping*, para lhe dar o estado atual de execução de uma tarefa específica. Quando o estado da tarefa atinge *DONE* são registados a hora e a data de conclusão, *Conclusion Time*, e é liberado o botão *Get Output* para que o utilizador possa obter todos os resultados.

- *Get Output*: O utilizador ao carregar neste botão está a dar ordem ao portal que deseja descarregar todos os ficheiros definidos no *OutputSandBox* da tarefa especificada. De momento o utilizador pode fazer o *download* do *OutputSandBox* de uma tarefa sempre que quiser (desde que não apague o registo da tarefa no portal), mas o objetivo é sempre que o utilizador faz *download* dos resultados, apagá-los do portal ou apagá-los ao fim de determinado tempo. Tal como acontece na infraestrutura *grid*, sempre que um utilizador obtém os resultados de uma tarefa esses serão posteriormente eliminados do *Computing Element* onde se encontram. Isto serve para que a capacidade de armazenamento dos computadores não exceda o limite ao fim de determinado tempo. Se todos os utilizadores deixarem os seus resultados indefinidamente nas máquinas, extravasavam com certeza a capacidade dos disco rígidos. De notar que os utilizadores do portal têm a capacidade de guardar todos os seus dados diretamente na infraestrutura *grid*, mas para isso precisam de acrescentar novos atributos ao seu ficheiro de definição de tarefa JDL (para guardar todo o *OutputSandBox* num SE *Storage Element* especificado no *OuputData*, para guardar cada ficheiro individualmente). Esses atributos podem ser acrescentados recorrendo ao botão *Add Element*, na página de definição de tarefas.

- *Cancel*: Este botão permite ao utilizador cancelar uma tarefa que se encontra em execução na infraestrutura *grid*. Após uma tarefa ser cancelada, esta poderá ser posteriormente submetida para execução.

- *Clear*: O botão *clear* apaga todo o ambiente criado para a tarefa do portal. Após o utilizador clicar neste botão o registo da tarefa, será apagado da fila de tarefas. Sendo assim será eliminada a entrada na base de dados e também todos os diretórios e ficheiros criados.

Como podemos ver, este serviço fornece toda uma visão, para o utilizador, de tudo o que está a decorrer na infraestrutura em relação as suas próprias tarefas submetidas. Com este, o utilizador poderá ter à sua disposição informações sobre as suas tarefas e também tem a capacidade de efetuar ações sobre as mesmas. De notar que para todas as ações

é necessário um certificado *proxy* válido, como explicado no ponto *submit* do parágrafo anterior. Só o botão *clear* é que não necessita do certificado *proxy* válido, visto que só interage com o servidor onde se encontra alojado o portal e não com a infraestrutura *grid*.

4.5 Conclusão

Este capítulo, serviu essencialmente, para levantar os requisitos necessários para a construção do portal Grid UP, para descrever as tecnologias disponíveis e utilizadas e também para descrever todas as funcionalidades existentes atualmente no portal.

Em relação aos requisitos, foram descritos os tipos de utilizadores que poderão utilizar o portal, físicos, engenheiros, cientistas de computadores. Foram também descritos os tipos de aplicações a que o portal Grid UP se deve reger e também as opções de *design* que este deve apresentar como, por exemplo, construir uma interface simples e intuitiva. Em relação às tecnologias enumeradas na secção 4.3 todas elas foram utilizadas na construção do portal. Todas essas tecnologias têm uma particularidade em comum – são todas *open source*. O facto dessas serem *open source* permite à universidade fazer uma "espécie de publicidade" a essas tecnologias o que permite mostrar que é possível fazer boas aplicações sem ter que pagar quantidades enormes de dinheiro para a aquisição de determinado *software* ou tecnologia. Por último, foram descritas todas as funcionalidades suportadas atualmente pelo portal Grid UP. Essas funcionalidades permitem a qualquer utilizador submeter e gerir todas as suas tarefas individualmente.

Capítulo 5

Implementação

5.1 Introdução

Neste capítulo, descrevemos toda a arquitetura do portal *grid* da Universidade do Porto. Explicamos a função de todos os *portlets* criados para o portal, bem como toda a atividade inerente à execução dos mesmos como, por exemplo, tabelas de gestão de tarefas dos utilizadores, organização dos diretórios de tarefas dos utilizadores e acessos à infraestrutura *grid* através da biblioteca jLite.

A estrutura deste capítulo é bastante simples contendo somente a introdução, a arquitetura do portal Grid UP onde será mostrada a arquitetura utilizada na sua construção e por fim a conclusão.

5.2 Arquitetura do Portal Grid UP

A Figura 5.1 mostra a arquitetura do portal Grid UP. A arquitetura está dividida em duas partes principais, cliente e servidor. A parte cliente é responsável por mostrar todo o conteúdo e as funções existentes no portal. Em termos de processamento, o cliente não apresenta grandes responsabilidades visto que praticamente todo o processamento é efetuado no lado do servidor. No lado do servidor, implementamos seis *portlets* responsáveis por gerir/controlar todo o processo de submissão para o utilizador e também responsáveis por mostrar informação. Na camada mais alta utilizamos os *portlets* sobre Liferay e Tomcat 6.0. Na camada mais baixa, de gestão de recursos, temos as interfaces com a base de dados, sistema de ficheiros e com a biblioteca jLite.

Como menciomado no parágrafo anterior o servidor é responsável pela maior parte do processamento, compreende o servidor Tomcat, a base de dados, o sistema de ficheiros e também o ponto de acesso à infraestrutura *grid* da Universidade do Porto. A tecnologia utilizada para a construção do portal foi Liferay. Esta tecnologia traz consigo o servidor *web* Tomcat responsável pela disponibilização de conteúdo na *web*. Esta também é

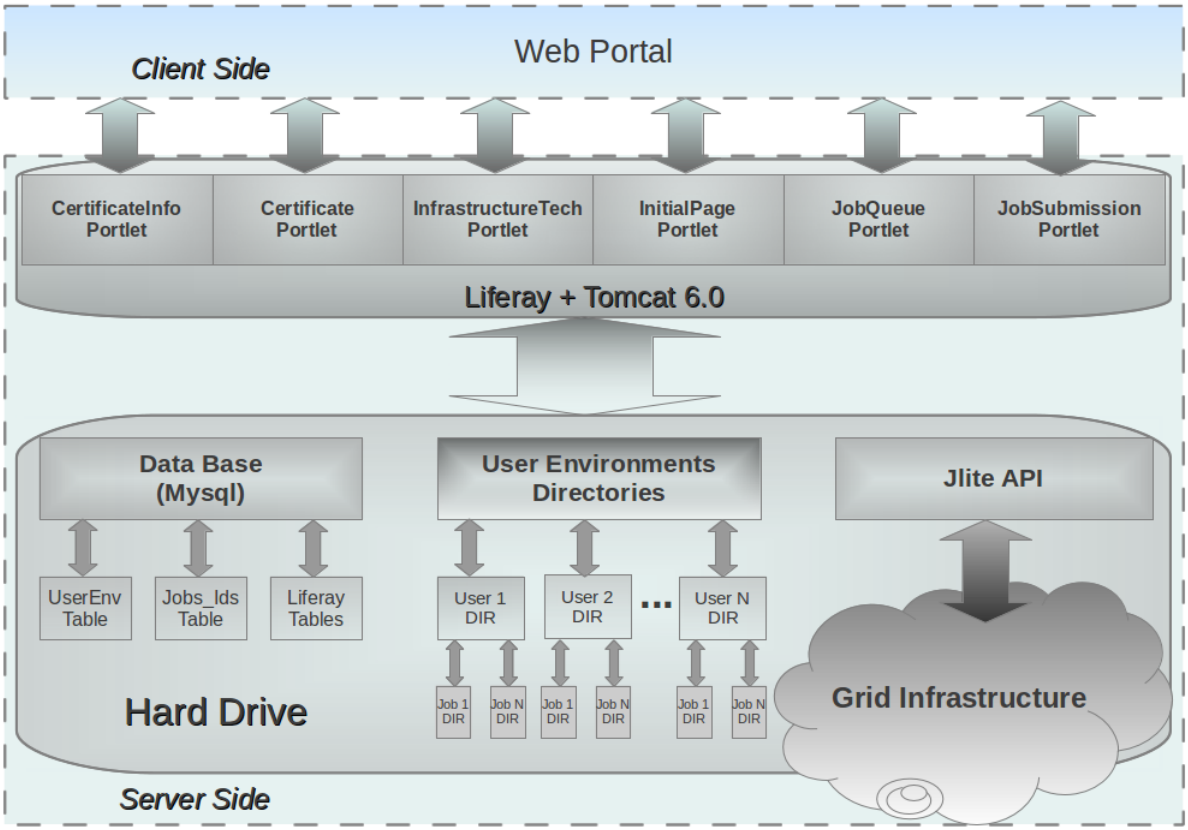

Figura 5.1: Arquitetura do portal *grid* da Universidade do Porto

responsável pela criação e por disponibilizar um conjunto de *portlets*.

Os *portlets* disponibilizados pelo servidor são essenciais para que o utilizador tenha todo um conjunto de serviços à sua disposição. Como se pode verificar na figura 5.1 o portal Grid UP disponibiliza um conjunto de *portlets* acessíveis a todos os utilizadores autenticados.

- CertificateInfo-Portlet: Este *portlet* é responsável só por mostrar informação sobre como pedir um certificado pessoal. Este indica ao utilizador o formulário a preencher, bem como o local onde este tem que se deslocar para confirmar a sua identidade. Ele é também responsável por instruir o utilizador de como exportar o seu certificado pessoal do seu navegador *web*. Ver Figura A.6 do Anexo A, lado direito.

- Certificate-Portlet: Este *portlet* é o serviço responsável pela gestão dos certificados pessoais. Cada utilizador tem o seu diretório pessoal independente dos restantes. Sendo assim, para cada utilizador é criado um diretório *.globus* onde é colocado o seu certificado pessoal. A função deste serviço é carregar um certificado pessoal para este mesmo diretório separando as chaves (pública e privada), caso necessário. De notar que o utilizador tem a capacidade de submeter as suas chaves separadamente, tendo ele mesmo a responsabilidade da as separar. Ver Figura A.6 do Anexo A.

- InfrastructureTech-Portlet: Este *portlet* mostra somente informação sobre toda a infraestrutura *grid* da Universidade do Porto, tal como a localização dos diferentes *clusters* e as entidades responsáveis. Ver Figura A.4 do Anexo A.

- InitialPage-Portlet: Este *portlet* disponibiliza alguma informação de apresentação. Atualmente mostra informação geral sobre o *middleware* gLite e sobre as suas componentes. Ver Figura A.3 do Anexo A.

- JobQueue-Portlet: Este *portlet* é o serviço responsável pela gestão de todas as tarefas do utilizador. Com este serviço o utilizador tem a capacidade de submeter, verificar o estado, cancelar e obter o resultado de uma determinada tarefa. Cada tarefa é gerida separadamente e tem um identificador único, não havendo assim conflito nem confusão entre elas. Sendo assim, cada tarefa tem um diretório separado onde são colocados todos os ficheiros que são carregados. Este também é o local onde são descarregados os resultados das tarefas. Sempre que um utilizador submete uma tarefa é criada uma entrada numa base de dados. Posteriormente essa entrada será lida por este serviço, isto para que possa apresentar a tarefa ao utilizador bem como ações que este pode efetuar sobre a mesma. Além das ações, para cada tarefa, este serviço, fornece também alguma informação adicional tal como o nome da tarefa definida pelo utilizador, o *id* da tarefa (*string* aleatória), a data de submissão e a data de conclusão da tarefa. Ver Figura A.18 do Anexo A.

- JobSubmission-Portlet: Este *portlet* tal como o nome indica é o serviço responsável pela submissão de tarefas. Como mencionado no ponto anterior cada tarefa tem o seu diretório separado. Sendo assim, sempre que o utilizador define uma nova tarefa, este serviço é responsável por criar um novo diretório. O nome do diretório

criado é definido pelo utilizador sempre que ele define um nome para a sua tarefa, ou seja, nome do diretório é igual ao nome da tarefa. De notar que o utilizador só pode ter um nome único para cada tarefa. No caso do utilizador entrar com um nome duplicado, o portal emite um erro dizendo que já existe uma tarefa com esse nome. Este serviço, além de criar um diretório, cria também uma entrada na base de dados para essa tarefa, onde ficará registada a progressão da mesma ao longo do tempo. Este serviço tem também a capacidade de notificar o utilizador sobre o seu certificado pessoal, ou seja, informa o utilizador no caso deste querer submeter uma tarefa sem ter o certificado pessoal instalado. Se este for o caso, é apresentada uma mensagem de erro sobre a necessidade de submeter um certificado e redireciona o utilizador para uma página de informação e para o local onde poderá obter o certificado. Ver Figura A.10 do Anexo A

Os *portlets* descritos, para poderem funcionar corretamente, necessitam de todo um conjunto de operações realizadas a nível do disco rígido. Essas operações estão subdivididas em três categorias principais – base de dados, diretórios e a biblioteca jLite.

5.2.1 Base de dados (Mysql)

A base de dados é responsável por guardar informação relevante que poderá vir a ser reutilizada num futuro próximo. Essa informação é especialmente importante para manter o estado de todo o portal e também para gerir todo um conjunto de aplicações de forma coerente e correta. O portal Liferay contém de raiz algumas tabelas na base de dados como, por exemplo, para gerir os utilizadores, gerir permissões, guardar informação definida pelo administrador, entre outras. Todas essas tabelas são importantes, mas para o portal Grid UP, as mais importantes são aquelas que foram criadas para gerir todos os diretórios dos utilizadores e também das tarefas correspondentes.

Para a construção do portal Grid UP foram criadas duas tabelas, que são responsáveis por gerir todas as tarefas de cada utilizador de forma independente. Essas tabelas são *User_Env* e *Jobs_Id*. A Tabela 5.1 apresenta uma das tabelas criadas para tal efeito.

Tabela 5.1: Tabela User_Env

Value	Type Value
*user_id	*INT*
home_dir	*VARCHAR(100)*
tmp_dir	*VARCHAR(100)*
globus_dir	*VARCHAR(100)*
proxy_path	*VARCHAR(200)*
cert_dir	*VARCHAR(200)*
p_key_dir	*VARCHAR(200)*

A tabela 5.1 é composta por sete atributos e é responsável por definir todo o ambiente de

trabalho necessário a cada utilizador. De seguida, é então, explicada de forma detalhada a função de cada atributo:

- *user_id*: Este elemento permite identificar um utilizador na tabela *User_Env*. Este *id* é gerado quando o utilizador/administrador cria uma nova conta no portal Grid UP e é guardado na tabela de gestão de utilizadores do portal Liferay. O *id* que identifica um utilizador é um número do tipo *long*, o que permite inúmeros utilizadores diferentes no portal. Este valor é acedido primeiramente através de um pedido ao próprio *portlet* que retorna o *id* do utilizador corrente que está a utilizar o portal. De notar que este *id* é único (gerado automaticamente pelo Liferay), não havendo conflitos entre diferentes utilizadores.

- *home_dir*: Este elemento é responsável por guardar o diretório *home* de cada utilizador. O elemento *home_dir* tem o seguinte formato: *<path_to_here>/UserEnvironments/<user_id>*. Sendo assim, cada diretório é criado com o *id* de cada utilizador. Neste diretório *home* está localizado todo o ambiente de execução para cada utilizador diferente.

- *tmp_dir*: Este elemento é responsável por guardar a localização do diretório *tmp*. Este é um diretório temporário onde são guardados todos os ficheiros de uma tarefa antes de ser carregada, ou seja, este serve para guardar o ambiente de uma tarefa de forma temporária. Quando o utilizador define o nome para determinada tarefa, o conteúdo deste diretório é transferido para o novo diretório criado (com o nome da tarefa). O elemento *tmp_dir* tem o seguinte formato: *<path_to_here>/UserEnvironments/<user_id>/tmp*.

- *globus_dir*: Este elemento é responsável por guardar a localização do diretório onde se encontra o certificado pessoal de cada utilizador. Este é um diretório com permissões especiais, visto que contém informação importante – certificado pessoal. O formato deste diretório é o seguinte: *<path_to_here> /UserEnvironments/<user_id>/.globus*.

- *proxy_path*: Este elemento é responsável por guardar o caminho para o certificado *proxy* criado. Este certificado é necessário para quando o utilizador necessita de submeter tarefas ou de aceder a qualquer recurso na infraestrutura *grid*. Este diretório é importante para que o portal localize rapidamente cada certificado *proxy*. O formato deste caminho é o seguinte: *<path_to_here>/UserEnvironments/<user_id>/<proxy_certifico*

- *cert_dir*: Este elemento é responsável por guardar o caminho para a chave pública de cada utilizador. Esta chave está guardada no diretório *.globus*. Como cada *portlet* funciona individualmente, logo o espaço de endereçamento de memória é diferente para todos. Sendo assim é necessário ter um meio de acesso comum onde todos os *portlets* possam aceder a este tipo de informação. O formato deste caminho é o seguinte: *<path_to_here>UserEnvironments/<user_id>/<proxy_certificate>/.globus/<public_key>.pem*.

- *p_key_dir*: Este elemento é exatamente igual ao anterior mas ao invés de guardar a chave pública guarda a chave privada do utilizador. O formato deste caminho é o seguinte: *<path_to_here>/ UserEnvironments/ <user_id>/ <proxy_certificate>/ .globus/<private_key>.pem.*

Os elementos acima referenciados são os que compõem a tabela *User_Env*. Todos os elementos acima descritos são inicializados uma única vez quando os utilizadores acedem pela primeira vez o portal Grid UP. A chave primária da tabela *User_Env* é *User_id*, visto que este identificador é único para cada utilizador, ou seja, não existem utilizadores com *ids* iguais.

A Tabela 5.2 mostra os atributos necessários para a gestão de tarefas no portal Grid UP.

Tabela 5.2: Tabela Jobs_Ids

Value	Type Value
*user_id	*INT*
*job_name	*VARCHAR(100)*
job_id	*VARCHAR(100)*
submission_time	*DATETIME*
conclusion_time	*DATETIME*
dir_path	*VARCHAR(200)*
output_path	*VARCHAR(200)*
get_output	*TINYINT*
state	*ENUM(...)*

A tabela 5.2 é composta por nove atributos e é responsável pela criação do ambiente necessário para cada tarefa. De seguida, é então, explicada de forma detalhada a função de cada atributo:

- *user_id*: Como explicado na tabela anterior este elemento é responsável por identificar o utilizador na tabela *Jobs_Ids*. Este elemento se existir nesta tabela terá de existir obrigatoriamente na tabela *User_Env*, visto que esta só é inicializada/atualizada quando o utilizador se autenticar no portal e inicializar a tabela anterior.

- *job_name*: Este elemento representa o nome da tarefa atribuída pelo utilizador quando a sua submissão. O elemento *job_name* é definido pelo utilizador quando este carrega a tarefa para a fila de tarefas. Sempre que este define um novo nome para uma nova tarefa é criada uma entrada na tabela *Jobs_Ids* com o *id* do utilizador corrente e também é registado na mesma linha da tabela o nome definido para a tarefa. Para cada utilizador não poderá existir entradas *job_name* iguais, mas para utilizadores diferentes isso já é possível. Por exemplo, dois utilizadores podem ter definido o mesmo nome para determinada tarefa. De notar que o utilizador é avisado para alterar o nome se este já o tiver definido.

- *job_id*: Este elemento representa o *id* gerado pela infraestrutura *grid* para uma determinada tarefa. Este elemento é atualizado na tabela sempre que um utilizador submete uma tarefa para a infraestrutura. O elemento *job_name* é necessário para que os utilizadores possam identificar as suas tarefas mais facilmente, visto que o *id* é uma *string* longa de carateres aleatórios.

- *submission_time*: Este elemento representa a data e a hora de submissão de determinada tarefa. Este elemento é atualizado juntamente com o elemento *job_id* sempre que uma tarefa é submetida para a infraestrutura. O relógio utilizado para marcar a data e a hora é do servidor onde se encontra o portal Grid UP alojado.

- *conclusion_time*: Este elemento representa a data e a hora de conclusão de determinada tarefa. Este elemento é atualizado sempre que uma tarefa chega a um estado de finalização, como *DONE* ou *ABORTED*. Tal como o elemento anterior, o relógio de marcação de data e hora é a do servidor hospedeiro.

- *dir_path*: Este elemento representa o diretório de uma tarefa, visto que cada uma tem um diretório separado. O caminho para este diretório resulta da concatenação do diretório *home* do utilizador corrente com o nome da tarefa *job_name*. Sendo assim, este caminho tem o seguinte formato: *<path_to_here>/ UserEnvironments/<user_id>/<job_name>*. De notar que todo o ambiente de todas as tarefas de determinado utilizador estão contidas no diretório *home* do mesmo. A atualização deste elemento é efetuada quando o utilizador carrega a tarefa para a fila de tarefas, tal como acontece com *user_id*, e *job_name*.

- *output_dir*: Este elemento representa o diretório onde irão ser guardados os resultados de determinada tarefa. O caminho deste diretório resulta da concatenação do diretório da tarefa com o *id* da tarefa. Sendo assim, este apresenta o formato seguinte: *<path_to_here>/ UserEnvironments/<user_id>/<job_name>/<job_short_id>*. A atualização deste elemento é feita somente quando o utilizador deseja descarregar os resultados das tarefas para a máquina local.

- *get_output*: Este elemento é um número *booleano* que representa se o utilizador descarregou os resultados, de determinada tarefa, para a sua máquina ou não. A necessidade deste elemento recai sobre a confirmação a ter se o utilizador descarregou os resultados ou não. Sempre que o utilizador pede os resultados à infraestrutura *grid*, depois de descarregados esses serão imediatamente apagados. Com o portal Grid UP o utilizador poderá descarregar os resultados para o servidor e não para a sua máquina local, então este elemento permite-nos saber se eles foram ou não descarregados para a sua máquina local. Isto permite poupar conexões com a infraestrutura *grid*, visto que estas têm um *delay* elevado. Depois de descarregados os resultados, estes serão imediatamente eliminados do servidor, isto por razões de gestão de espaço em disco. Assim sendo, este elemento é atualizado sempre que o utilizador descarrega os resultados para a sua máquina local.

- *state*: Este elemento permite ao portal saber o estado atual da tarefa na infraestrutura *grid*. Sendo assim o portal é capaz de apresentar uma interface diferente para cada tarefa, dependendo do seu estado atual. Este elemento é atualizado sempre

que existem alterações de estado de uma tarefa, que vai desde o carregamento deste para o portal até à sua conclusão.

Os elementos acima referenciados vão sendo atualizados de acordo com o estado atual de determinada tarefa ou ação efetuada sobre a mesma pelo utilizador. Isto permite ao portal determinar os estados das tarefas e ajustar a interface (fila de tarefas) sempre que existe alterações de estado. Este ajuste de interface oferece ao utilizador um *feedback* simples e de fácil compreensão sobre todas as etapas do ciclo de vida de cada tarefa. A chave primária da tabela *Jobs_Ids* é feita pela junção de dois atributos *user_id* com *job_name*. Isto permite que para cada utilizador só exista uma tarefa com o mesmo nome. O código SQL capaz de realizar tal ação é o seguinte:

- **CONSTRAINT pk_idName PRIMARY KEY (user_id, job_name)**

5.2.2 Diretório *User Environments*

Como dito anteriormente, todos os utilizadores do portal *grid* da Universidade do Porto têm um ambiente de trabalho separado de todos os outros. Sendo assim, sempre que um novo utilizador acede ao portal, este cria um novo diretório com o *id* do utilizador corrente. Para cada diretório criado, são criados outros dois novos diretórios, dentro do anterior (filhos). A Figura 5.2 ilustra a árvore de diretórios para um utilizador. E esses diretórios são:

- Diretório onde serão guardados todos os ficheiros temporários na criação de uma tarefa. Como por exemplo o ficheiro JDL, os ficheiros de input etc.
- Diretório onde serão armazenadas as chaves pública e privada do utilizador em questão.

Os diretórios acima são criados logo na primeira autenticação do utilizador, mantendo-se assim para sempre desde que o utilizador não seja eliminado do portal. Além destes anteriores o portal também guarda o certificado *proxy* dentro do diretório *home* do utilizador em questão. Sendo assim sempre que o utilizador necessite de realizar alguma ação na infraestrutura *grid*, o portal vai ler e verificar o ficheiro *proxy certificate* do utilizador. Ao ler este ficheiro, poderá ocorrer três tipos de ações diferentes:

- Se o ficheiro não existir, o portal vai criar um *proxy certificate*, bastando ao utilizador colocar a palavra chave de proteção da sua chave privada.
- Se o ficheiro existir, mas se não estiver válido, o portal realiza a mesma operação anterior.
- Se o ficheiro existir e estiver válido, o portal simplesmente prossegue com o pedido para a infraestrutura elaborado pelo utilizador. A validação do certificado é simplesmente verificar o prazo de validade do mesmo, já que estes ao serem criados normalmente, têm validade de doze horas.

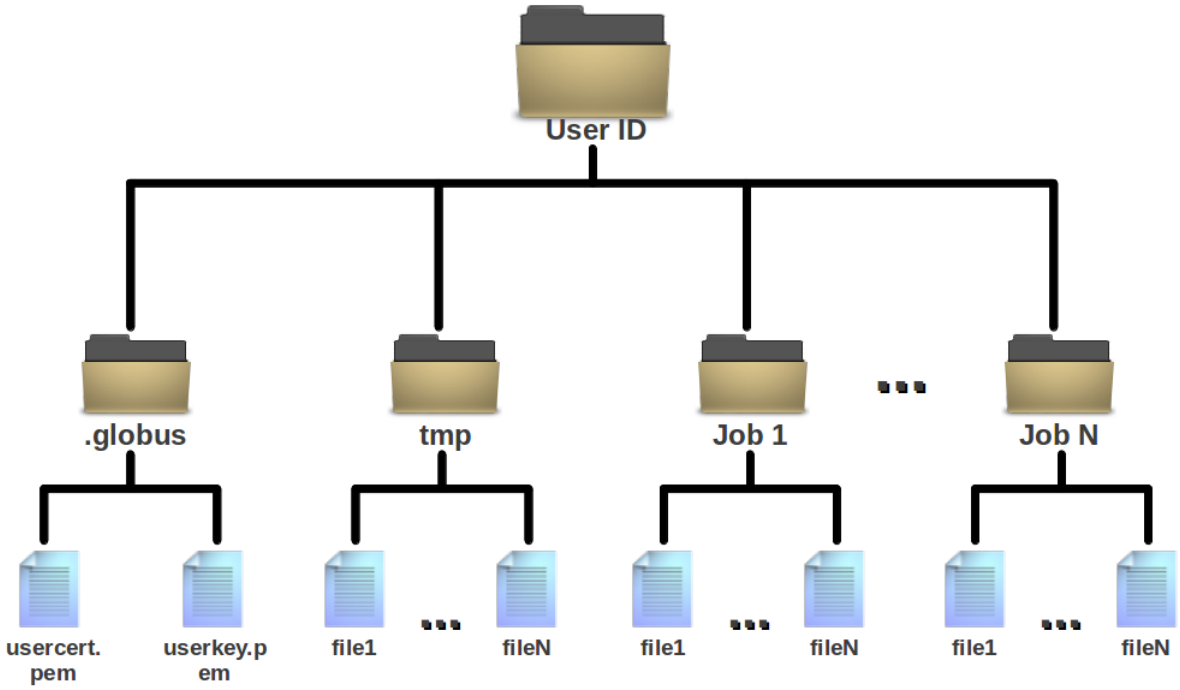

Figura 5.2: Árvore de diretórios *home* de um utilizador

Em relação às tarefas, cada tarefa tem um diretório criado separadamente. Sendo assim se um utilizador criar N tarefas, dentro do diretório *home* do utilizador existirá pelo menos N diretórios em que cada um irá representar uma tarefa diferente. Dentro de cada diretório de tarefa temos todos os ficheiros necessários para a sua submissão para a infraestrutura *grid*, Figura 5.2. Por exemplo, se um utilizador desejar executar um programa que tenha dependências a nível de ficheiros de entrada, o diretório terá de ter o seguinte:

- O ficheiro JDL criado pelo portal, responsável pela definição da tarefa. De notar que este ficheiro poderá ser carregado ou criado através do portal Grid UP.
- O programa executável, que deverá ser carregado através do portal pelo utilizador.
- Todos os ficheiros de entrada necessários para a execução do programa. De notar que todos esses ficheiros terão de ser carregados via portal individualmente. Para o caso dos ficheiros estarem alojados num servidor de armazenamento dentro da infraestrutura, o utilizador terá de definir o caminho para os mesmos no formulário JDL.

Sempre que uma tarefa termina a sua execução e que o utilizador deseje descarregar os resultados, o portal irá criar um diretório, dentro do diretório da tarefa, para guardar os resultados para posteriormente estes serem descarregados para a máquina local. O processo de obtenção de resultados passa por três fases principais:

- 1) O portal cria um novo diretório para armazenar os resultados e descarrega-os para esse diretório criado. O descarregamento em questão é realizado entre o WMS,

onde está armazenado o *OutputSandBox*, e o servidor onde se encontra alojado o portal.

- 2) No fim dos resultados descarregados o portal vai comprimir a pasta, dando origem a um arquivo *.zip*. A escolha recaída sobre este formato de compressão prende-se com o fato de ser multi-plataforma (Windows, Linux).

- 3) O portal vai criar um *Download Resourcer* para descarregar os resultados para a máquina local do utilizador. A descarga é feita através do gestor de *downloads* do navegador *web*, perguntando ao utilizador se deseja descarregar ou não os resultados.

Para concluir e resumir esta secção, para cada utilizador existirá um diretório que contém todos os ambientes das suas próprias tarefas, juntamente com uma pasta temporária, uma pasta com o certificado pessoal e por último o certificado *proxy* necessário para delegação de direitos à infraestrutura.

5.2.3 jLite

A biblioteca jLite é responsável por encaminhar os pedidos vindos dos utilizadores do portal Grid UP para a infraestrutura *grid* existente na Universidade do Porto. Resumidamente, esta biblioteca serve como meio de comunicação entre o portal e toda a infraestrutura *grid*. Esta biblioteca não necessita de ter nenhuma UI "User Interface" instalada e é multi-plataforma. A configuração da biblioteca jLite para estar acessível a uma infraestrutura é bastante simples, bastando efetuar quatro passos essenciais:

- Criar um diretório para certificados confiáveis de uma ou mais autoridades certificadoras. Esses certificados servem para provar que a autoridade certificadora que assinou determinado certificado pessoal tem autorização para tal (ver `http://www.igtf.net/` para descarregar os certificados). O caminho para tal diretório é o seguinte: *jLite/etc/certs/ca*.

- Criar um ficheiro que defina a autoridade certificadora responsável. Este ficheiro é necessário para que a biblioteca consiga associar a autoridade certificadora às chaves definidas no ponto anterior. O caminho para tal ficheiro é o seguinte: *jLite/etc/certs/vom/vo.up.tp/voms.up.pt.lsc*.

- Criar um ficheiro que defina a localização do servidor VOMS, para determinada organização virtual. Este é necessário para que a biblioteca consiga criar o certificado *proxy* para o utilizador. O caminho para tal ficheiro é o seguinte: *jLite/etc/vomses/vo.up.pt*, no caso de estarmos a utilizar a VO da Universidade do Porto.

- Criar um ficheiro que defina a localização do servidor WMS. A localização é definida através de um URI "Uniform Resource Identifier" e este ficheiro serve para definir o servidor responsável pela gestão e distribuição de todas as tarefas na infraestrutura *grid*. O caminho para tal ficheiro é o seguinte: *jLite/etc/wms/vo.up.pt/glite_wms.conf*.

A configuração dos ficheiros acima referidos necessita somente de uma linha de configuração, tornando-os assim simples e rápidos de criar. Sempre que o utilizador descarrega a biblioteca jLite para a sua máquina local, os ficheiros necessários já vêm previamente criados e configurados bastando assim ao administrador alterar alguns parâmetros para ajustar ao seu ambiente. Sendo assim, a configuração torna-se ainda mais simples e rápida. De notar que, para cada organização virtual, existem ficheiros/diretórios com o nome dela - *vo.up.pt*. Se o administrador desejar inserir uma nova organização virtual terá de criar esses ficheiros/diretórios como o nome dela. Outro ponto a realçar é que os certificados contidos no diretório *jLite/etc/certs/ca*, terão de ser atualizados frequentemente, isto por questões de segurança.

Após a configuração acima referenciada, a biblioteca jLite está pronta a utilizar. Esta, então, permite aos utilizadores gerir todo o ciclo de vida das tarefas de forma simples e também permite que este crie o seu certificado *proxy*. Sendo assim o programador tem a vida facilitada, bastando este definir algumas classes Java e gerir o necessário em alto nível sem ser necessário saber de detalhes de mais baixo nível como tratamento de erros, localização dos servidores, *output*, respostas estruturadas, entre outras.

5.3 Conclusão

Neste capítulo foi descrita a arquitetura do portal *grid* da Universidade do Porto. O portal utiliza a tecnologia Liferay, juntamente com o servidor *web* Tomcat 6.0, para gestão de contas e permissões de utilizadores e também para disponibilizar todo o conteúdo *web*. Além disso, essa tecnologia é também responsável por criar e gerir um conjunto de *portlets*. Esses são essenciais para disponibilizar os serviços necessários para gestão e criação de tarefas no portal Grid UP. Esses serviços são ligados à infraestrutura *grid* através da biblioteca, escrita em Java, jLite. Essa biblioteca é fundamental para fazer a comunicação entre o portal Grid UP e toda a infraestrutura *grid*, pois esta permite ocultar bastantes detalhes técnicos. Além da simplicidade, esta também oferece portabilidade, fazendo com que se possam migrar os serviços para diferentes ambientes sem grandes problemas, visto que estamos a trabalhar com uma linguagem de programação multi plataforma.

Para cada utilizador é criado um ambiente de trabalho diferente dos demais, fazendo com que ele tenha todo o seu trabalho numa só pasta e organizado, permitindo assim resolver "conflitos" que possam existir. O mesmo é válido para as tarefas que também têm ambientes separados das restantes. Para cada utilizador existe uma pasta com todas as tarefas criadas de forma organizada e humanamente legível (nome explícito). Toda esta gestão hierárquica de diretórios é feita recorrendo a tabelas Mysql que são responsáveis por armazenar informação relevante e também por identificar o utilizador em questão, isto para que o portal possa organizar todo o ambiente corretamente (por exemplo, colocar as tarefas pertencentes a determinado utilizador no local correto).

Atualmente, o portal Grid UP ofecece aos utilizadores uma forma simples de submeter tarefas, desde que sejam do tipo *Normal*. O utilizador tem também a capacidade de gerir todo o ciclo de vida de uma tarefa, capacidade de verificar o estado atual de cada tarefa,

cancelar uma tarefa, voltar a submeter a mesma tarefa e obter por fim todos os resultados das tarefas. Além disso e para finalizar, o utilizador tem a possibilidade de fazer o pedido de um certificado pessoal e de submetê-lo via portal.

Capítulo 6

Conclusões e Trabalhos Futuros

A Universidade do Porto (UP) tem recursos computacionais que fazem parte da infraestrutura de *grid* Europeia, EGI. Atualmente esses recursos não estão a ser muito utilizados pela comunidade de investigadores UP. Para incentivar os utilizadores a utilizarem os recursos computacionais existentes, esta dissertação contribui com um portal *grid*. O portal Grid UP permite aos utilizadores executarem um conjunto de tarefas, graficamente.

O portal Grid UP é uma interface *web* construída recorrendo a tecnologias como Liferay, Java, ferramenta Vaadin e a biblioteca jLite. O portal Liferay é a tecnologia que permite disponibilizar todo o conteúdo na Internet bem como organizá-los através de *portlets*. Os *portlets* são componentes *web*, *pluggable*, que podem ser adicionados e geridos (*look-and-feel*) independentemente. Além disso, Liferay permite a um administrador gerir grupos de utilizadores bem como executar funções administrativas como correr *garbage collector*, definir o servidor de email, depuração, entre outros. A biblioteca Java jLite é a interface de comunicação entre o utilizador e a infraestrutura *grid*. Esta biblioteca permite aos utilizadores submeterem e gerir todo o ciclo de vida de uma tarefa. A ferramenta Vaadin permite ao programador criar todo um conjunto de páginas *web* somente utilizando a linguagem Java. A vantagem desta é a capacidade de gerar código eficiente, recorrendo à ferramenta da Google GWT. Outra vantagem é a facilidade de uso (cliente e servidor num só ficheiro) e também a capacidade de organização de código. Visto isto, o portal Grid UP permite ao utilizador submeter tarefas para a infraestrutura *grid* sem a necessidade de conhecimentos aprofundados sobre computação distribuída e infraestruturas de *grid*. No momento, este portal permite submeter tarefas do tipo Normal. Sendo assim, o utilizador é capaz de submeter tarefas para a infraestrutura, verificar os estados das suas tarefas, cancelá-las e resubmetê-las sem a necessidade de carregá-las novamente. Para utilizar o portal Grid UP, o utilizador tem que ter um certificado pessoal válido devidamente assinado por uma autoridade certificadora. Uma vez com o certificado pessoal, em sua posse, o utilizador pode submetê-lo via portal para poder ter acesso a todas os serviços disponibilizados.

O desenho do portal foi feito com base no desenho de outros portais disponíveis tais como o GSG (Gisela Science Gateway). Desta forma, aproveitamos funcionalidades que já existiam e adaptamos para os nosso propósitos. Por exemplo, decidimos por apresentar

um *layout* em níveis, onde utilizadores menos experientes têm acesso a uma interface básica e utilizadores com mais experiência terão acesso a menus mais avançados. No momento, o portal implementa funcionalidades básicas necessárias para a submissão de tarefas simples (p.e. tarefas do tipo *Normal* do gLite). O utilizador poderia submeter qualquer tipo de tarefas, a partir de um JDL já criado bastando apenas fazer pequenas modificações, na interface. Escolhemos, num primeiro momento disponibilizar o portal com os requisitos básicos e, a partir das opiniões dos utilizadores, fazer modificações.

Os objetivos iniciais foram praticamente todos cumpridos, à excepção de alguns pontos que serão discriminados de seguida.

- O portal atende aos requisitos dos utilizadores, pois permite qualquer utilizador, pertencente à instituição, utilizar o portal. Em relação às tarefas o utilizador tem capacidade de poder computacional mas não de armazenamento, no momento.

- Um dos objetivos enunciados era aumentar a visibilidade do serviço *grid* da Univisidade do Porto. Só saberemos isso ao certo quando o portal estiver em produção, mas de momento este apresenta uma interface simples e intuitiva para que este objetivo seja alcançado com sucesso. Além disso o portal facilita imenso o uso de infraestruturas grid.

- O utilizador tem a capacidade de se autenticar. O tipo de autenticação usado (tradicional) ainda não é autenticação federada. Qualquer utilizador tem a capacidade de criar uma nova conta se o desejar, posteriormente este terá que fazer a confirmação de *email*. A palavra chave do utilizador é gerada automaticamente logo o utilizador terá que colocar um *email* válido para ter acesso ao portal.

- O portal oferece ao utilizador a capacidade de fazer o pedido e submeter o certificado pessoal. O utilizador para pedir o certificado, atualmente, preenche um formulário web, envia-o e depois terá que se deslocar a uma autoridade certificadora/registo para provar a sua identidade. O pedido de certificados ainda se encontra um pouco burocrático, mas num futuro próximo este serviço vai tornar o processo de acesso à infraestrutura menos burocrático, isto é, o pedido de certificados será automatizado, oferecendo assim a possibilidade ao utilizador pedir o seu certificado pessoal no momento sem sair do lugar. De notar que o certificado ficará pronto a utilizar de seguida.

- O utilizador tem a possibilidade de criar as suas tarefas com total liberdade. A criação de tarefas é facilitado, pois o formulário já se encontra pré-preenchido, assim o utilizador poderá criar qualquer tarefa muito rapidamente. De notar que as tarefas suportadas até ao momento são do tipo *Normal*. Ao preencher o formulário o utilizador tem também à sua disposição campos de ajuda que têm com objetivo a compreensão do mesmo. Nestes campos de ajuda é facultado também um exemplo real de utilização do campo em questão. Um outro objetivo destes campos é fazer com que o utilizador não sinta a necessidade de procurar informação sobre o campo na *web*, fazendo com que a criação de tarefas seja um processo pouco moroso e simples.

- O portal oferece ao utilizador uma fila de tarefas gráfica, onde este pode gerir todo o ciclo de tarefas. Esta fila de tarefas é responsável por fornecer alguma informação importante e também por oferecer acções que o utilizador poderá realizar posteriormente. De notar também que o utilizador gere cada tarefa individualmente.

- Em relação à informação o portal fornece alguma informação sobre a localização dos recursos, caraterísticas técnicas dos recursos, entre outros. A página inicial oferece um breve resumo sobre computação *grid* e sobre o *middleware* gLite. Esta informação deverá ser expandida, por exemplo, passar noticías sobre a Universidade do Porto, sobre todas as novidades de computação *grid* e alguma informação relacionada e importante sobre computação, principalmente.

Como trabalho futuro planeamos introduzir as seguintes funcionalidades no portal, para aumentar a sua usabilidade e a qualidade dos serviços:

- Autenticação federada. Os utilizadores deverão poder se autenticar no portal Grid UP sem a necessidade de criação de novas credenciais, mas apenas através das credenciais *sigarra*.

- Pedido de certificados automáticos. Os utilizadores deverão poder obter o seu certificado pessoal através de autenticação federada. Através de autenticação federada o utilizador pode provar a sua identidade, permitindo assim diminuir a burocracia no pedido e gestão de certificados pessoais.

- Integração de ferramentas gráficas para a criação de tarefas. Essas ferramentas gráficas são chamadas de *workflows*, permitindo assim ao utilizador definir uma tarefa graficamente.

- Extensão a outros tipos de tarefas. Permitir ao utilizador submeter tarefas MPI, Parametric, Collection e DAG. Teoricamente, o utilizador pode carregar qualquer tipo de tarefa, mas, da forma como a interface está implementada, mostrando o JDL total ao utilizador para inspeção, ainda não suporta todos os tipos de tarefas.

- Integração com Logical File Catalog (LFN). Permite que o utilizador possa gerir ficheiros na *grid*, como por exemplo, criar réplicas , carregar e descarregar ficheiros para/a *grid*.

- Oferecer a possibilidade ao utilizador de selecionar a organização virtual que deseja utilizar.

- Fornecer a possibilidade ao utilizador de criar *templates* de tarefas que foi produzindo.

- Criar uma maneira simples do utilizador definir os atributos *Requirements* e *Rank*, para poder facilitar o uso dos mesmos.

Apêndice A

Casos de Utilização

A.1 Exportar Certificados Pessoais do Navegador *Web*

Como mencionado nos capítulos anteriores todos os utilizadores para utilizarem o portal *grid* da Universidade do porto necessitam de um certificado pessoal para que possam utilizar todos os recursos disponíveis na infraestrutura *grid*. Então no fim de um determinado utilizador preencher o formulário de pedido do certificado pessoal (disponível no portal Grid UP), de assinar um documento necessário (disponível no portal Grid UP) e de entregá-lo juntamente com a identificação pessoal numa autoridade de registo (por exemplo, reitoria da Universidade do Porto) ou autoridade certificadora (por exemplo, LIP Lisboa), este irá receber um email da autoridade certificadora para instalar o seu certificado no navegador *web*. Esta secção mostra como é que um utilizador exporta o seu certificado pessoal do seu navegador *web* para a sua máquina pessoal, de notar que o navegador utilizado é o Mozilla Firefox. Quando o utilizador se encontra na tab *Advanced* do navegador e posteriormente na tab *Encryption* este deve clicar no botão *View Certificates*, Figura A.1, parte A. Este botão irá abrir uma nova página (*Certificate Manager*) que contém todos os certificados pessoais, as autoridades certificadoras etc. Na tab *Your Certificates* o utilizador deve escolher o certificado com o nome LIPCA (autoridade certificadora reconhecida pela infraestrutura *grid* UP) e clicar no botão *Backup* para exportá-lo, Figura A.1, parte B.

Quando o utilizador clica no botão *Backup*, este terá de escolher um nome para o certificado e também a localização para onde esse será descarregado. Depois disso será pedido ao utilizador que introduza uma palavra chave de segurança para proteger o certificado pessoal, Figura A.2.

A.2 Página Inicial

Na página inicial do portal Grid UP é apresentada uma breve descrição sobre o que é computação *grid*, a composição da infraestrutura *grid* e sobre o *middleware* gLite, ver

Figura A.3. Além da página inicial, existe uma outra página acessível responsável por mostrar a localização dos *clusters* de infraestrutura *grid* da Universidade do Porto e também por mostrar algumas das suas características, Figura A.4. Como se pode ver na Figura A.3 e A.4, só existem duas página acessíveis, isto porque o utilizador ainda não se autenticou no portal. As páginas acessíveis são somente páginas de informação, página inicial e página com a informação sobre a infraestrutura. Enquanto o utilizador não se autenticar no portal, os serviços existentes para interação com toda a infraestrutura não irão aparecer. Sendo assim, o utilizador ver-se-á obrigado a autenticar-se no portal tanto para utilizar os serviços quanto para visualizá-los.

Para se autenticar no portal o utilizador tem de clicar no *link sign-in* no canto superior direito, Figura A.3, para que apareça uma página onde este possa colocar o seu endereço de email e a respetiva palavra chave, Figura A.5. A autenticação é realizada num novo *portlet*, substituindo todos *portlets* existentes na página corrente. Quando um utilizador se autentica com sucesso, o conteúdo da página volta a estar como inicialmente. Após a autenticação o utilizador já tem ao seu dispor todos os serviços para a interação com a infraestrutura de *grid*.

A.3 Gestão de Certificados

Nesta secção, mostramos como é que os utilizadores submetem o seu certificado pessoal através do portal Grid UP. Sem o certificado válido o utilizador não poderá executar nenhuma ação sobre a infraestrutura *grid*. Na Figura A.6 é mostrada a página inicial do serviço *Upload Certificate*. Este serviço é responsável por submeter o certificado pessoal, bem como instruir o utilizador a fazer o pedido do mesmo. A parte direita, desta mesma imagem, informa o utilizador sobre os procedimentos a fazer caso este não tenha certificado - *Request a new Certificate*. Além dessa informação, este contém também informação sobre os passos de como exportar os certificados pessoais dos diferentes navegadores - *Export a certificate from your browser*. A parte esquerda é responsável pela submissão do certificado e também pelas instruções de como separar a chave pública e privada manualmente através do ambiente *shell* Linux.

Para submeter um certificado, o utilizador tem que clicar no botão *Browse...* e selecionar o certificado exportado pelo navegador, neste caso o certificado chama-se *usercert.p12*, Figura A.7. Quando o certificado estiver selecionado, este deverá clicar no botão *Upload Now* para ser dada a ordem de carregamento do certificado para o portal. O utilizador só poderá selecionar ficheiros com a extensão *.p12* e *.pem* e estes estão também restritos a um tamanho máximo, cerca de um MegaByte. Se o certificado selecionado tiver a extensão *.p12* o portal irá separar automaticamente a chave pública e privada. Assim o utilizador só necessita de introduzir a palavra chave de exportação e definir a palavra chave para a sua chave privada, Figura A.8.

No fim do certificado selecionado e das palavras chaves definidas o portal mostrará que o certificado foi devidamente carregado e também irá dizer que se encontra o certificado instalado no portal, Figura A.9. De notar que o utilizador só necessita submeter o

certificado pessoal uma única vez e tem também a liberdade de removê-lo e de submetê-lo novamente. Como os certificados pessoais têm um prazo de validade, os utilizadores poderão ter a necessidade de remover o certificado antigo, substituindo assim por um novo.

A.4 Submissão de Tarefas

Nesta secção, mostramos como é que um utilizador pode submeter tarefas para a infra-estrutura *grid*. Mais uma vez é de frisar que todos os utilizadores que desejem utilizar este serviço terão a necessidade de submeter o seu certificado pessoal via portal. A Figura A.10, mostra a página inicial do serviço *Submit Job*. Este serviço é sub-dividido em duas partes. A primeira parte (esquerda) é onde os utilizadores poderão preencher o formulário, definindo assim a sua tarefa. De notar que o formulário está com alguns atributos já preenchidos, permitindo assim diminuir o trabalho aos utilizadores. Para cada atributo definido no formulário, existe um botão de ajuda que é responsável por explicar o mesmo e dar um exemplo. A ajuda é ativada simplesmente com o passar do rato por cima da imagem de ajuda. Já na segunda parte (direita) é onde os utilizadores poderão submeter todos os ficheiros necessários para execução e definição da sua tarefa. No máximo, o utilizador poderá submeter até dez MegaBytes de ficheiros. Este valor é o valor imposto pelo *middleware* gLite para submeter ficheiros via *User Interface*. Para ficheiros da maior dimensão o utilizador precisa carregá-los para um *Storage Element* e definir o seu URI, por exemplo, defini-lo (URI) no atributo *InputData*.

Como já foi dito em capítulos anteriores, o utilizador tem a capacidade de submeter um ficheiro JDL já pré-definido. Um exemplo disso pode ser visto na Figura A.11, em que o utilizador submete um ficheiro chamado *hello.jdl* e o portal preenche todo o formulário automaticamente com os atributos definidos no ficheiro JDL. O preenchimento dos atributos é feito para que o utilizador possa alterar algum campo, se necessário. Sendo assim, os utilizadores podem criar os seus *templates* e submetê-los sempre que necessário. De notar também que foi carregado outro ficheiro chamado *test.sh* que serve de argumento ao programa *sh* definido. Mais uma vez, todos os ficheiros necessários terão de ser obrigatoriamente submetidos, para que a tarefa termine a sua execução com sucesso.

Para os utilizadores mais avançados, existe um botão *Add Element* que permite a estes definirem atributos mais específicos, de uma lista de possíveis atributos definidos pela linguagem JDL, para a sua tarefa, Figura A.12. Aqui o utilizador seleciona o atributo que deseja, preenche os campos pedidos e adiciona-o ao formulário principal. Isto é importante para que estes tipos de utilizadores não estejam limitados aos atributos simples, que para estes muitas vezes são insuficientes. Além de adicionar atributos, o utilizador pode ver o ficheiro JDL carregando no botão *Show JDL File* e reiniciar a janela principal, botão *Reset Window*. A exibição do ficheiro JDL serve para mostrar aos utilizadores o ficheiro JDL com a definição da sua tarefa, Figura A.13. O estado da página é sempre mantido. Mesmo que o utilizador clique no botão atualizar do seu navegador, a página não será reiniciada. O botão *Reset Window* serve, então, para repor toda a página inicial e os

valores por defeito.

No fim do utilizador definir toda a sua tarefa e carregar todos os ficheiros necessários, este está apto, então, para carregar a tarefa para o portal. Essa função é feita através de um *click* no botão *Load Job* que é responsável por criar todo o ambiente necessário à tarefa, como por exemplo, registá-la numa base de dados e criar o diretório separado. Como mostra a Figura A.14, é pedido que o utilizador defina um nome para a tarefa. Ao submeter uma tarefa a infraestrutura gera um *id* para a mesma, mas como este não é humanamente compreensível daí a necessidade do utilizador definir um nome para a sua própria tarefa. Para finalizar, se estiver tudo bem com a definição da tarefa o portal reinicia a janela para que o utilizador possa criar uma nova e mostra também uma mensagem de sucesso como pode ser visto na Figura A.15.

A.5 Fila de Tarefas

Nesta secção será descrito o funcionamento da fila de tarefas do portal Grid UP. A fila de tarefas é essencial para que os utilizadores possam verificar o estado das suas tarefas e também serve de registo. Na Figura A.16, é mostrada a forma como a fila de tarefas está organizada. A fila de tarefas consiste basicamente numa tabela em que cada linha corresponde a tarefas diferentes. Para cada tarefa, é mostrada alguma informação sobre a mesma, tal como:

- O nome da tarefa (*JOB NAME*). Este nome é definido pelo utilizador, para que este possa distinguir as diferentes tarefas mais facilmente.
- O *id* da tarefa (*JOB ID*). Este *id* é gerado quando o utilizador submete a tarefa para a infraestrutura *grid*.
- Data e hora de submissão da tarefa (*SUBMISSION TIME*). Este campo regista a data e a hora de quando a tarefa foi submetida. Este registo é feito quando o utilizador submete a tarefa para a infraestrutura.
- Data e hora de conclusão da tarefa (*CONCLUSION TIME*). Este campo, tal como o anterior, regista a data e a hora mas de conclusão da tarefa. Este registo é feito quando a tarefa termina a sua execução, ou seja, quando atingir o estado *DONE*.
- Estado da tarefa (*STATUS*). Este mostra o estado atual da tarefa e é alterado sempre que houver modificações no seu estado.

Além da informação referenciada acima, cada linha da tarefa apresenta também alguns botões, que são responsáveis por efetuar operações na infraestrutura *grid*. Os botões representam bem a sua função, não havendo necessidade para grandes explicações. Por ordem (da esquerda para a direita) e função deles é: submissão de tarefas para a infraestrutura, atualização do estado da tarefa, obter os resultados após a execução da tarefa, cancelar a execução da tarefa e por fim remover o registo da tarefa do servidor. Existem alguns

botões que estão bloqueados e outros ativos, isto para que o utilizador seja conduzido automaticamente para a etapa seguinte sem a necessidade deste se enganar e também deste saber qual o passo seguinte. A Figura A.16, mostra alguma informação *null*. Esta informação irá ser preenchida automaticamente, à medida que o utilizador vai avançando com a tarefa.

A primeira vez que o utilizador tenta submeter uma tarefa ser-lhe-á pedido uma palavra chave para a criação do certificado *proxy*, Figura A.17 (*Password Window*). Essa palavra chave é a chave de proteção da sua chave privada, que terá de ser introduzida para que o certificado *proxy* possa ser criado com sucesso. Após introduzir a palavra chave, o utilizador, nas próximas doze horas, tem a liberdade de efetuar todo o conjunto de operações sem ter a necessidade de a introduzir novamente. Por exemplo, para submeter a primeira tarefa da tabela, chamada "*MyJob*", existe a necessidade de introduzir a palavra chave e para as restantes tarefas isso já não é necessário. De notar que o utilizador poderá sair (*logout*) as vezes que desejar do portal, porque enquanto o certificado *proxy* estiver válido, ele poderá efetuar ações sem nenhuma restrição.

Sempre que o utilizador efetua alguma ação na fila de tarefas o desenho da mesma é moldado dependendo das operações realizadas. Por exemplo, na Figura A.18, o utilizador submeteu três tarefas e o aspeto de cada linha foi alterado em relação à Figura A.16. Como se pode verificar a interface encaminha o utilizador para o passo seguinte, bloqueando as ações desnecessárias em cada etapa e também acrescentando informação nas colunas a *null*, conforme as etapas.

Quando uma tarefa termina a sua execução na infraestrutura *grid*, o botão *Get Output* é libertado para que o utilizador possa descarregar os resultados para a máquina local, Figura A.19. Sempre que um utilizador carrega nesse botão este está a dar a ordem ao servidor para descarregar os resultados da infraestrutura *grid* para o servidor que, posteriormente, irão ser comprimidos para que depois o utilizador final possa descarregá-los para a sua máquina local, utilizando o gestor de *downloads* do seu navegador *web*. Mais uma vez, é de realçar que a interface foi modificada quando a tarefa *host3* chegou ao seu estado final *DONE*, como se pode ver na Figura A.19.

A.6 Conclusão

Esta secção serviu essencialmente para mostrar o funcionamento do portal Grid UP e casos de uso. Foram mostradas imagens que ilustram os vários serviços existentes. Além das imagens apresentadas, foi também descrito o funcionamento de cada serviço separadamente.

Em suma, o portal Grid UP, no seu estado atual, permite aos utilizadores submeter e gerir tarefas de forma gráfica e também gerir o seu certificado pessoal. Só podem ser submetidas tarefas simples o que faz com que o portal ainda seja um pouco básico para alguns utilizadores mais avançados.

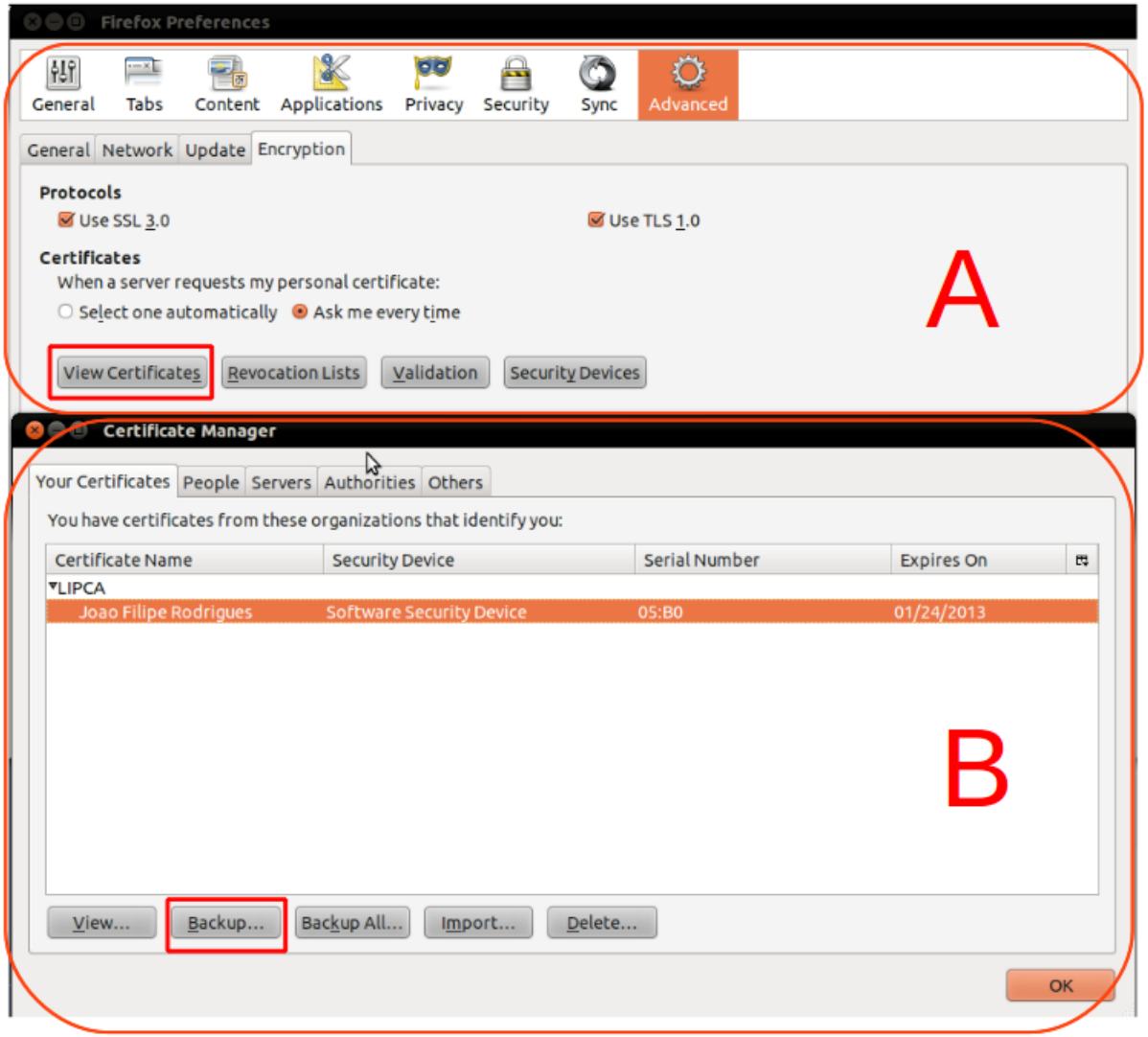

Figura A.1: Exportação do certificado pessoal - Backup.

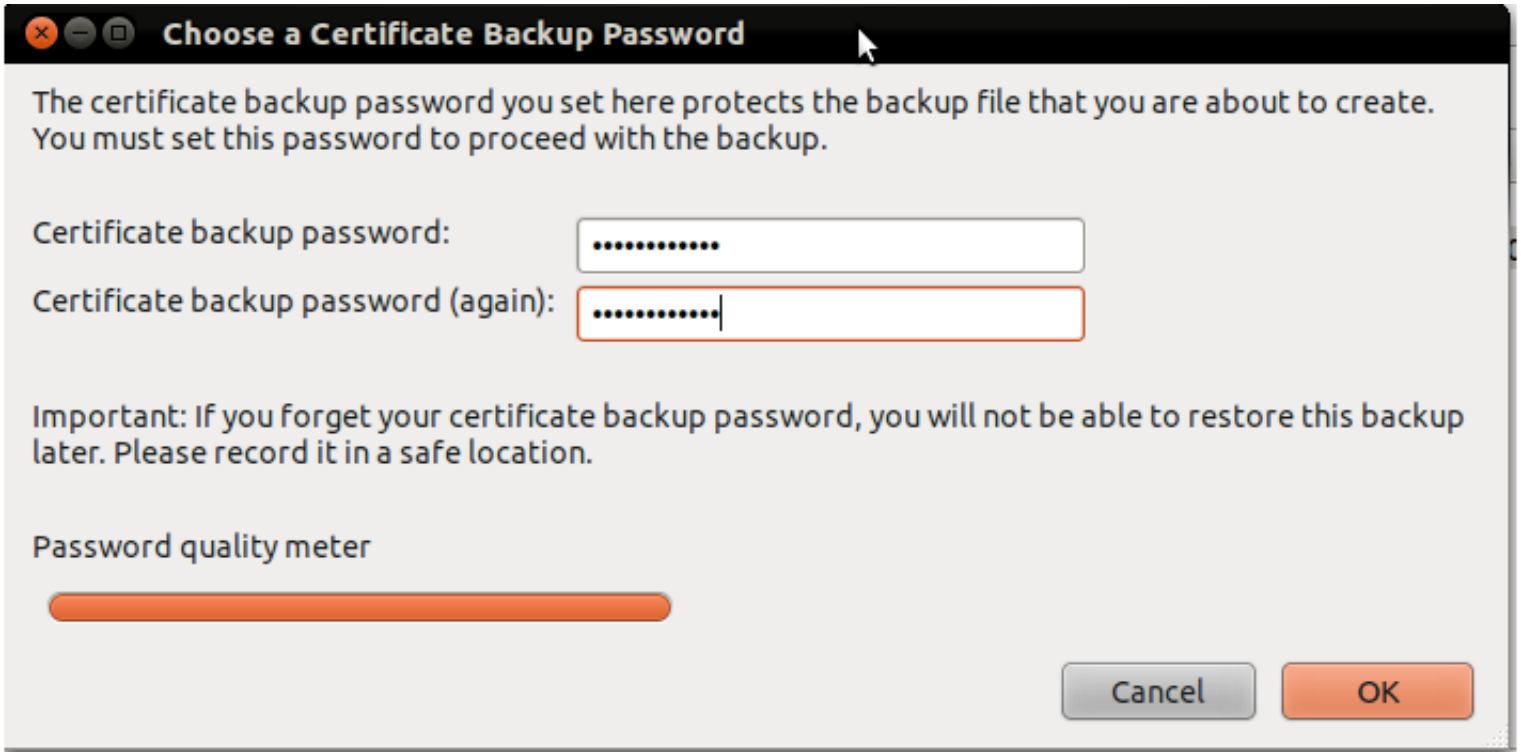

Figura A.2: Exportação do certificado pessoal - Palavra chave.

Figura A.3: Pagina Inicial.

Figura A.4: Pagina Inicial - Informação sobre os recursos Grid.

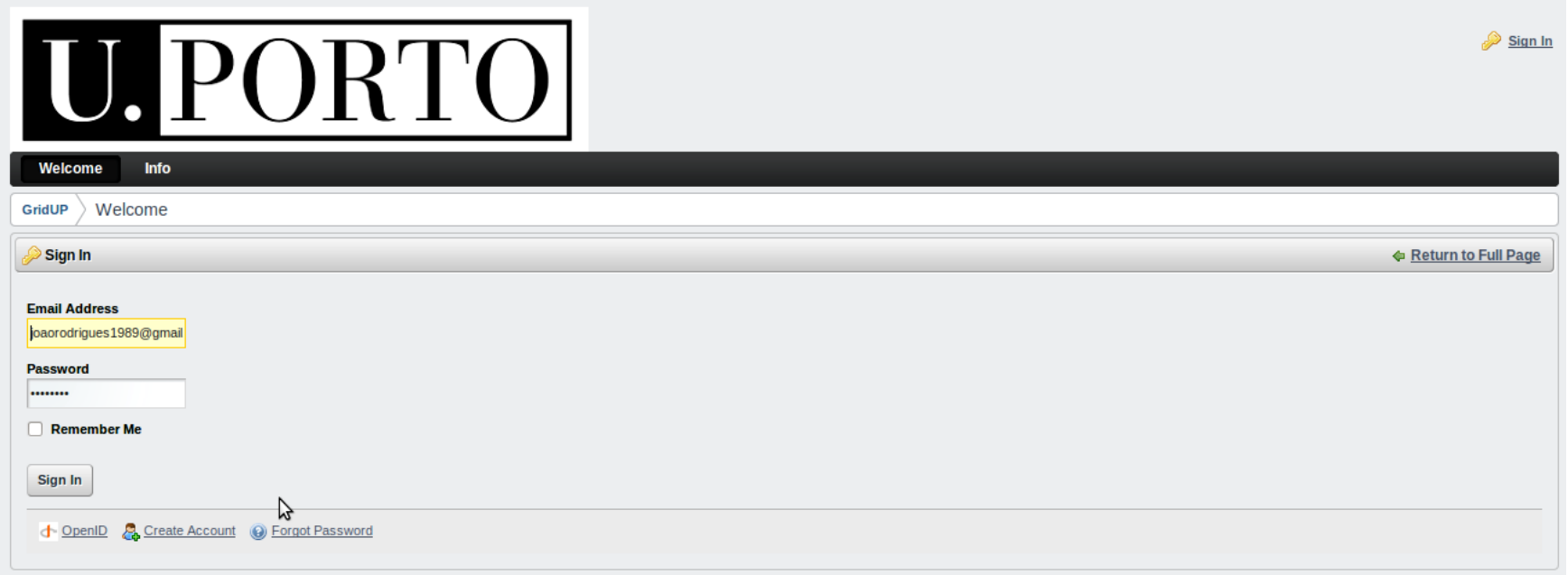

Figura A.5: Pagina de Autenticação.

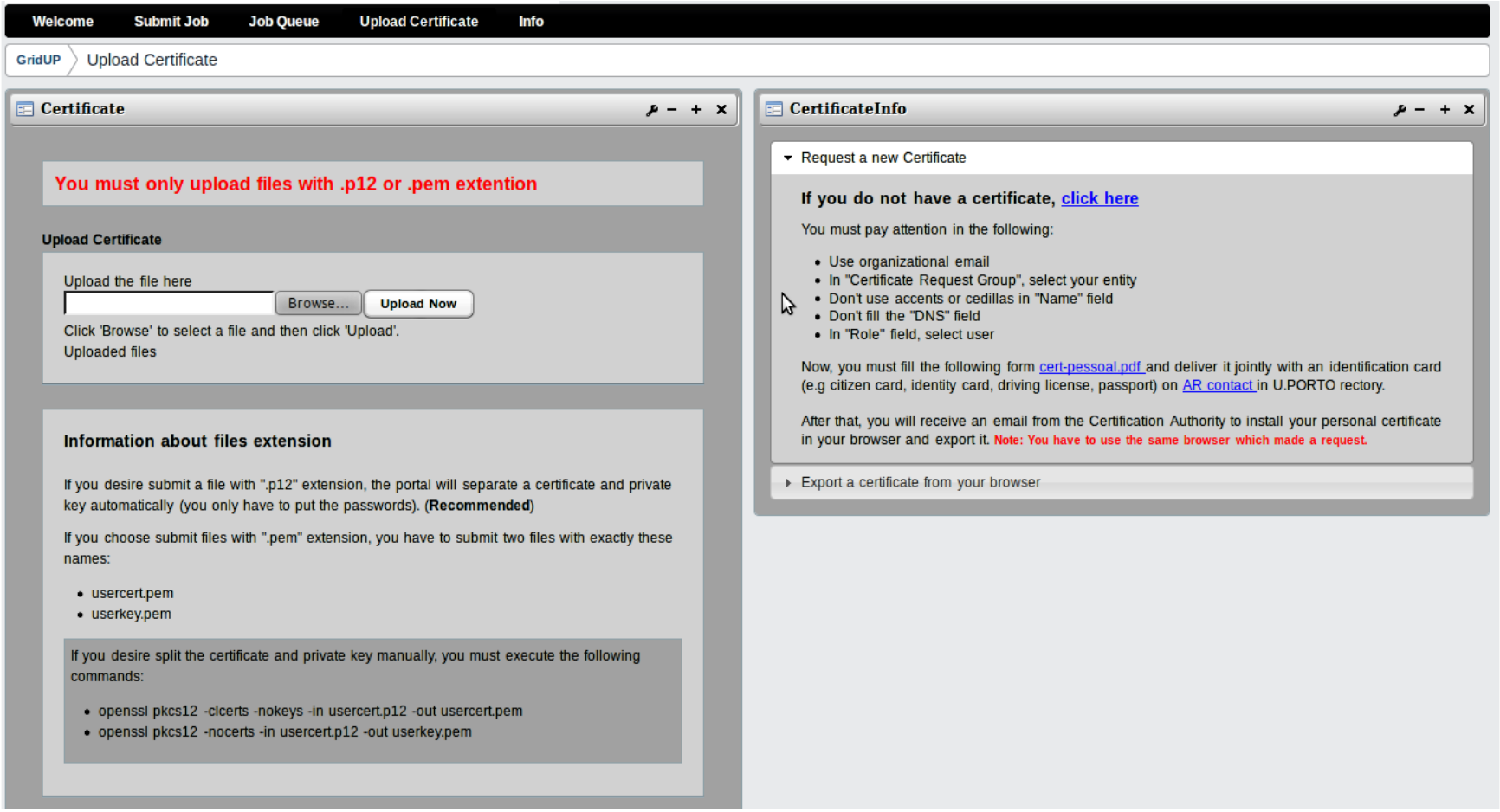

Figura A.6: Página inicial do serviço de gestão de certificados.

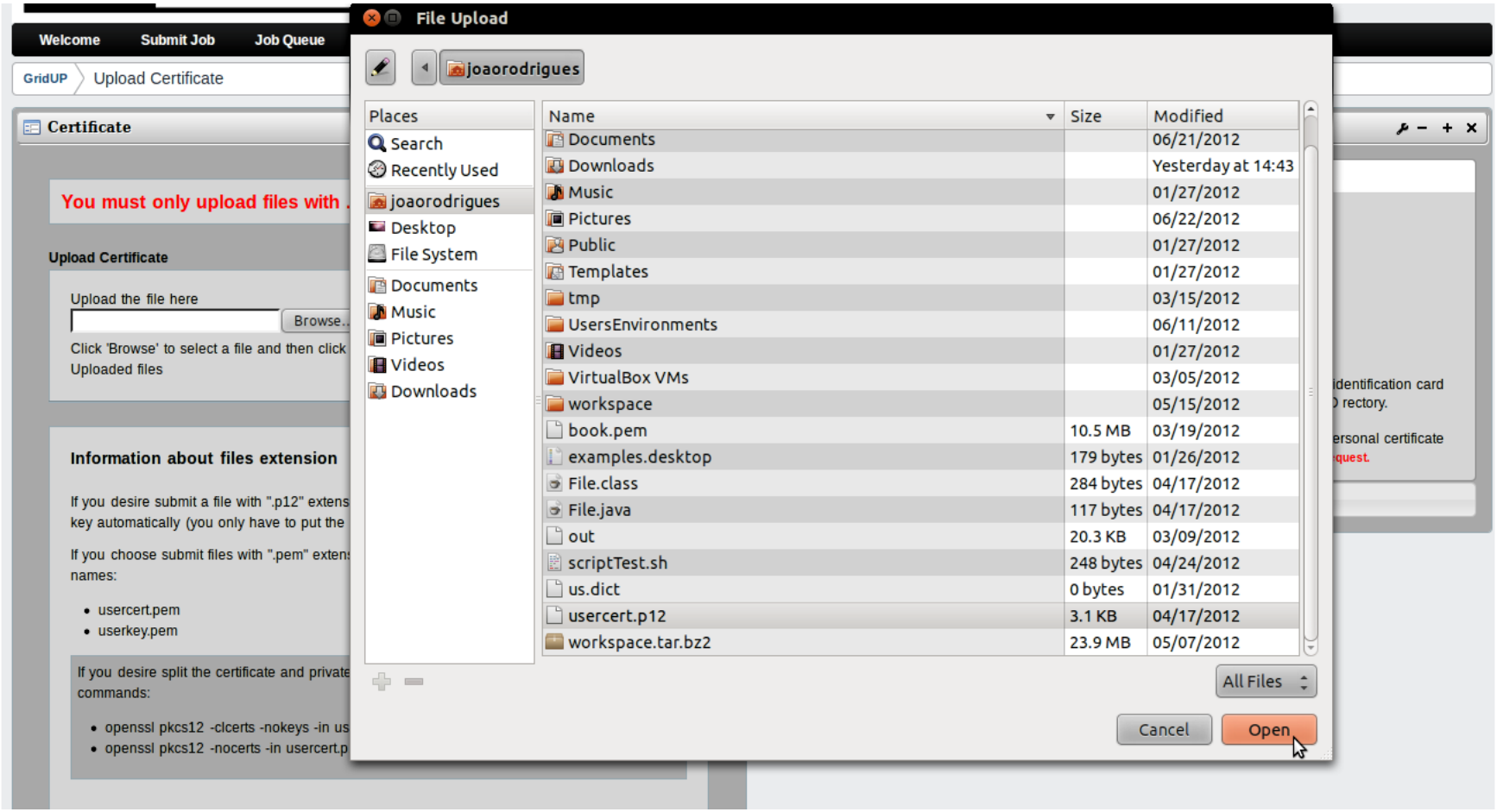

Figura A.7: Seleção do certificado pessoal.

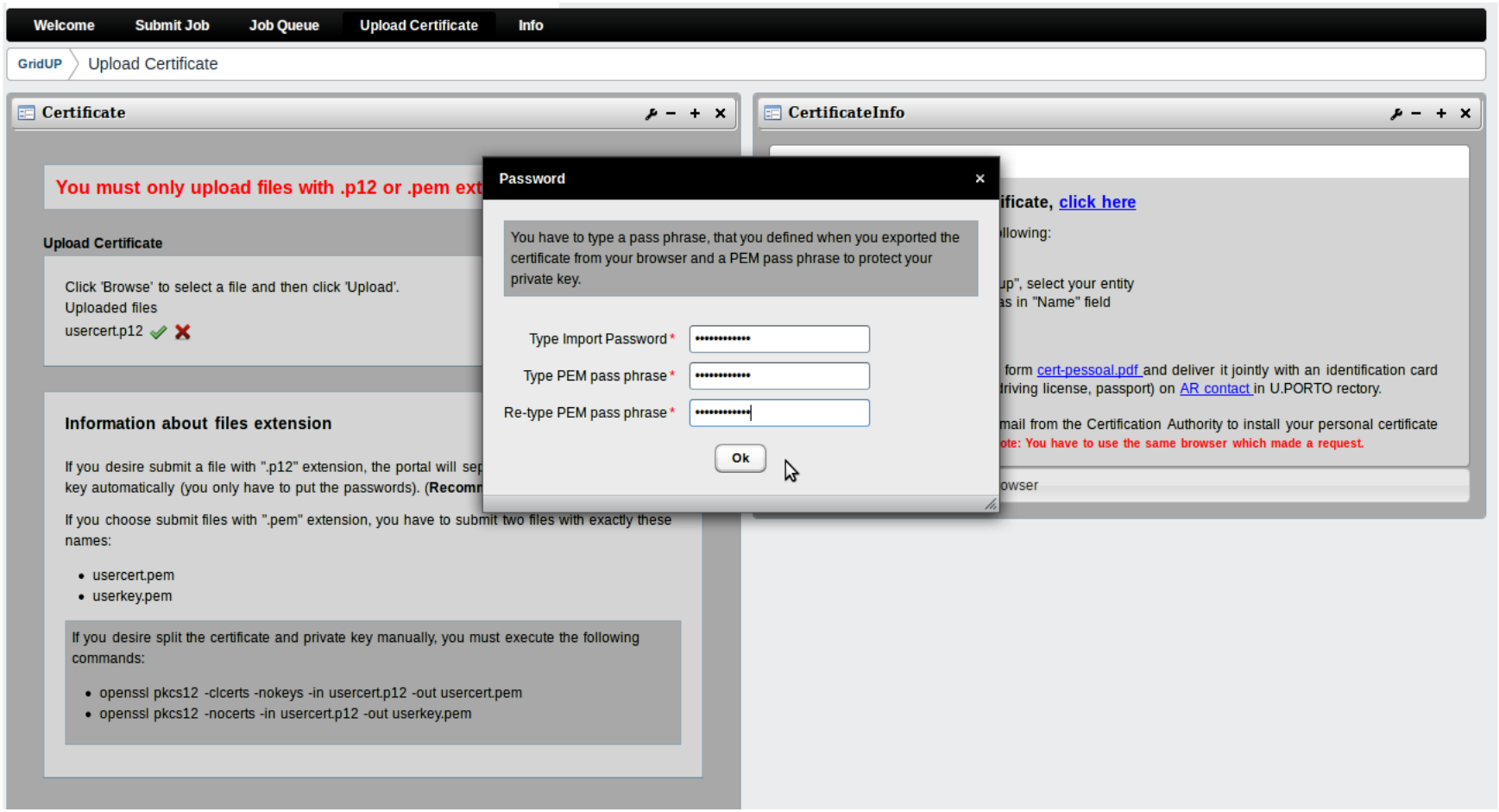

Figura A.8: Separação de chaves (pública e privada).

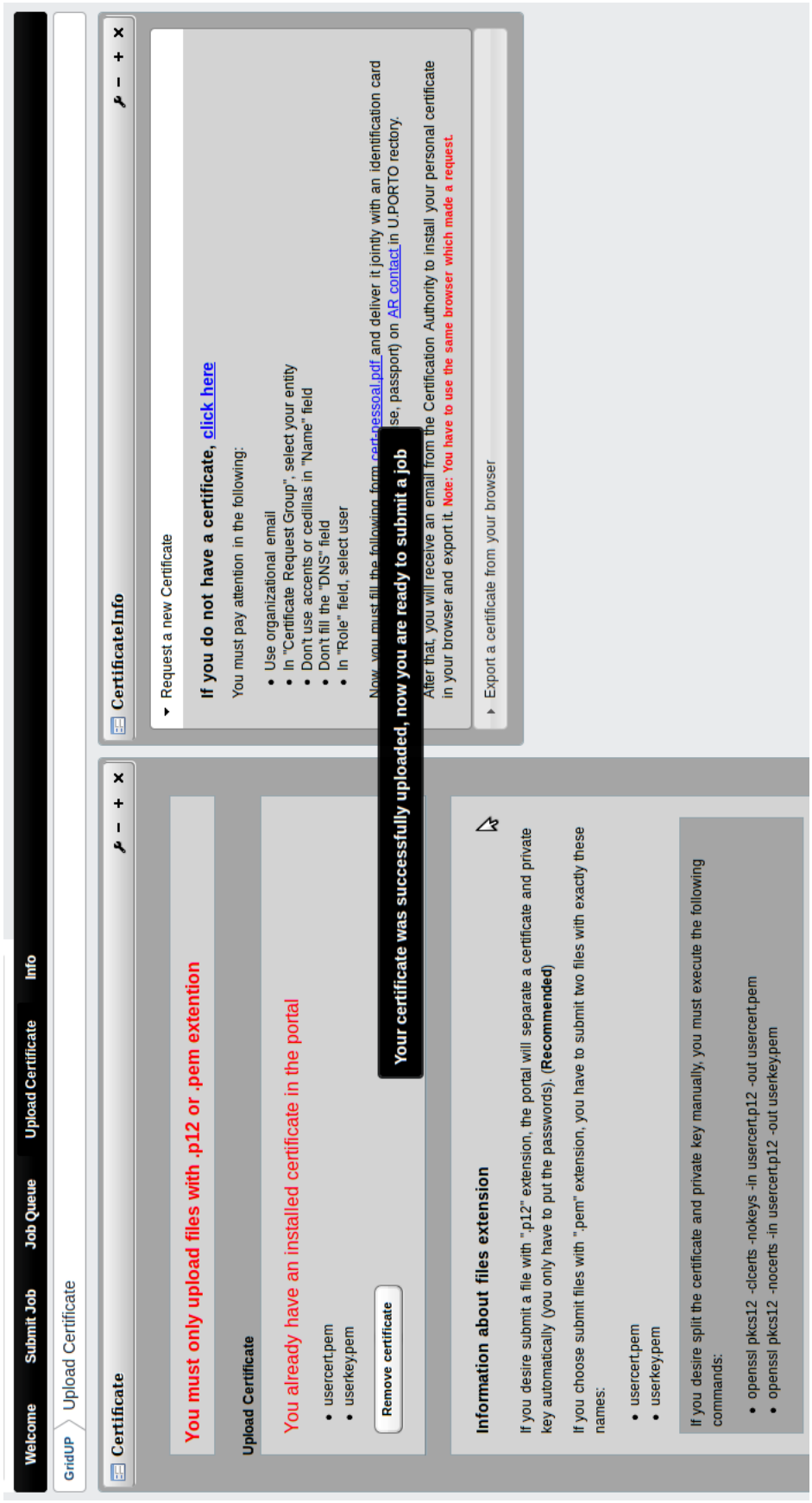

Figura A.9: Confirmação de instalação do certificado pessoal.

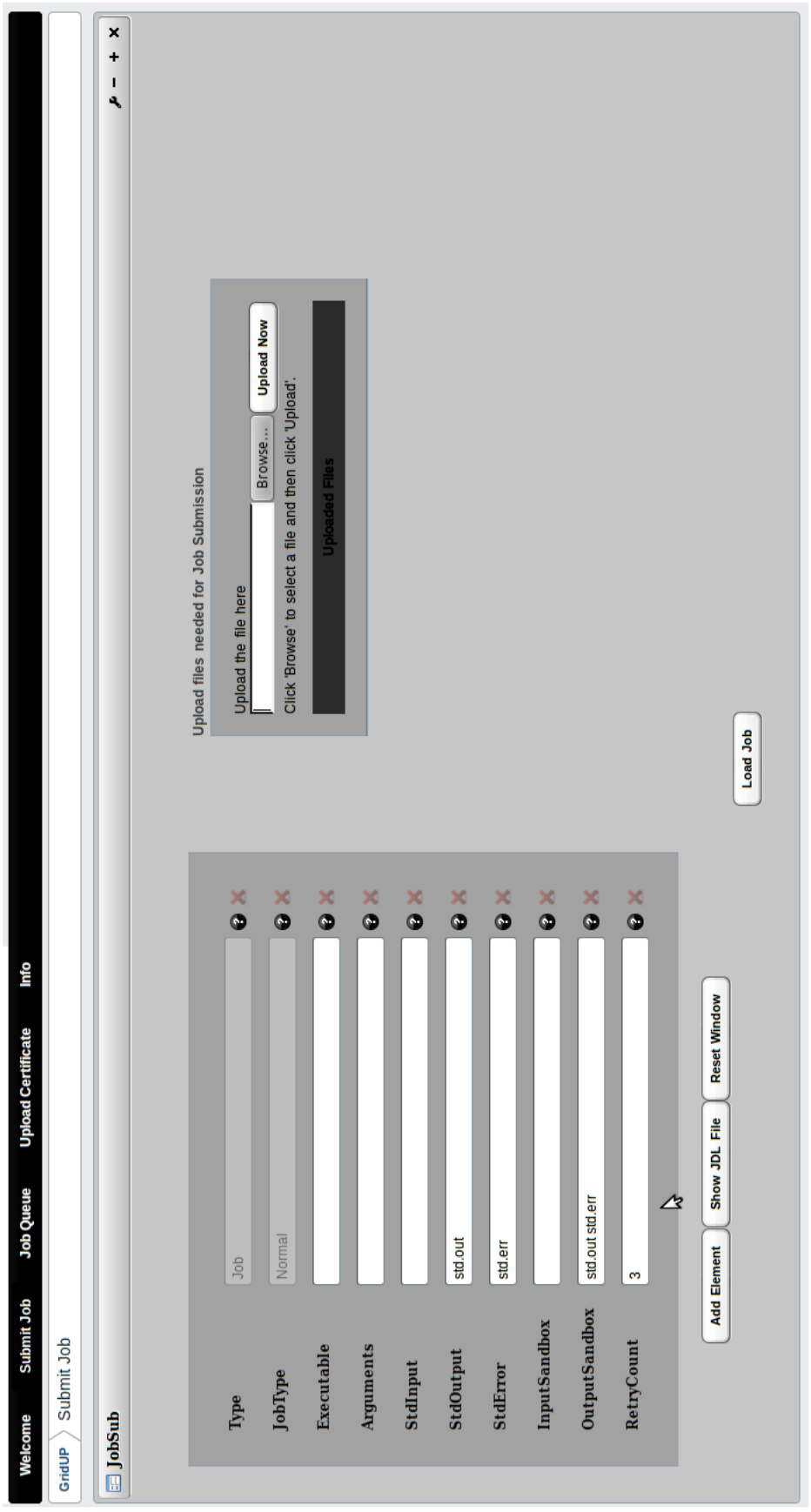

Figura A.10: Página inicial do serviço de submissão de tarefas

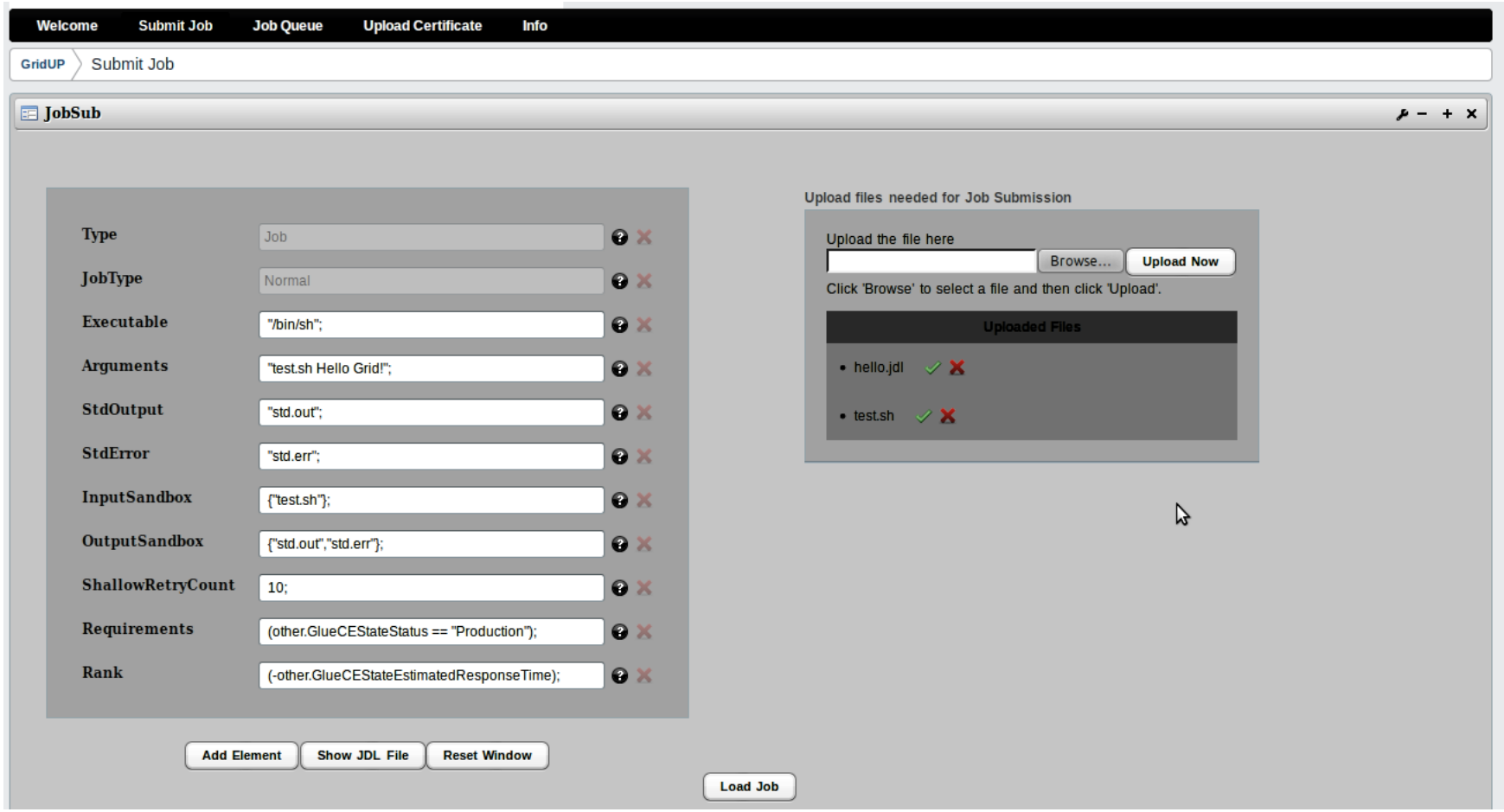

Figura A.11: Carregar um ficheiro JDL, préviamente definido

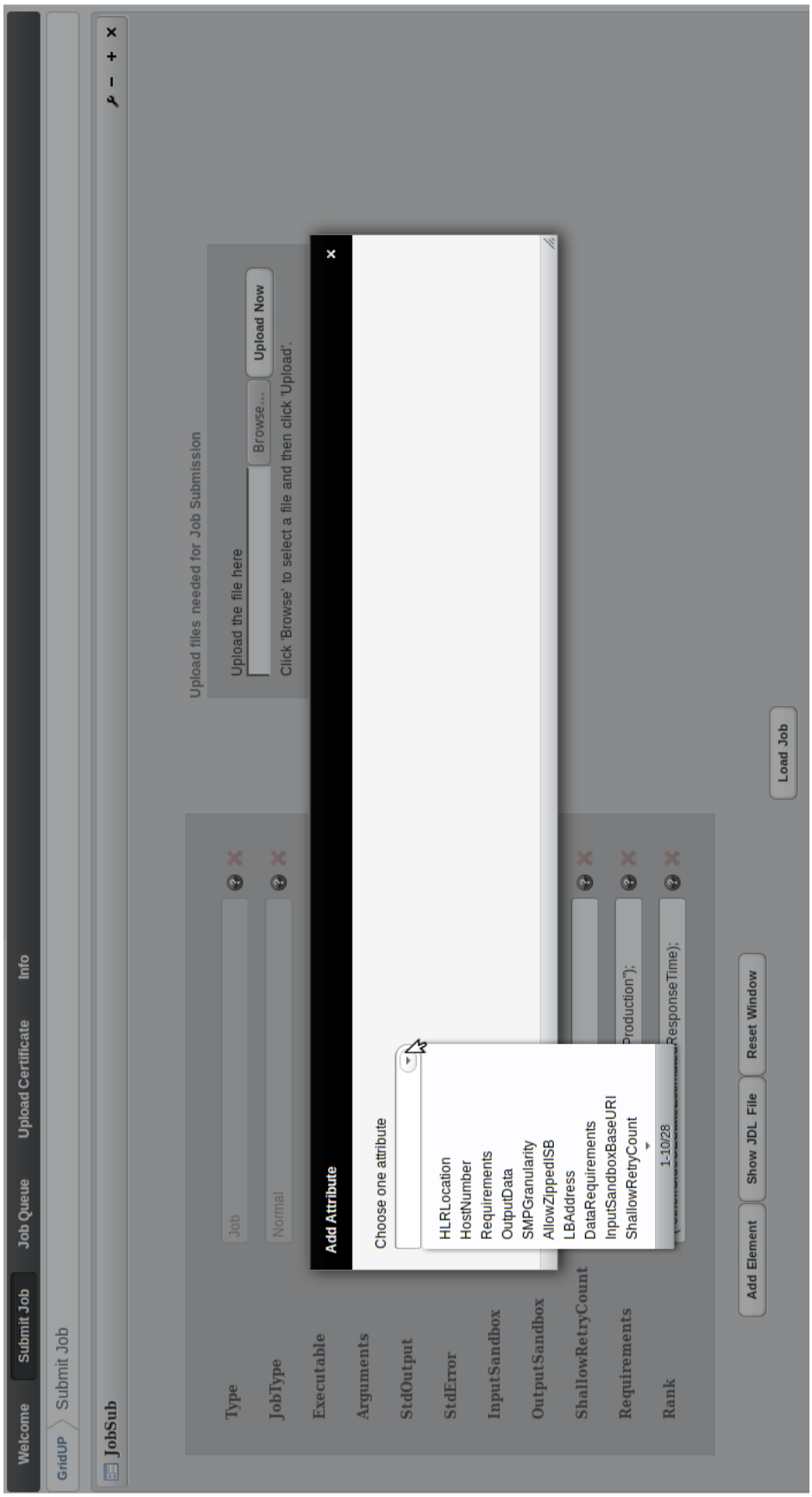

Figura A.12: Adicionar atributos

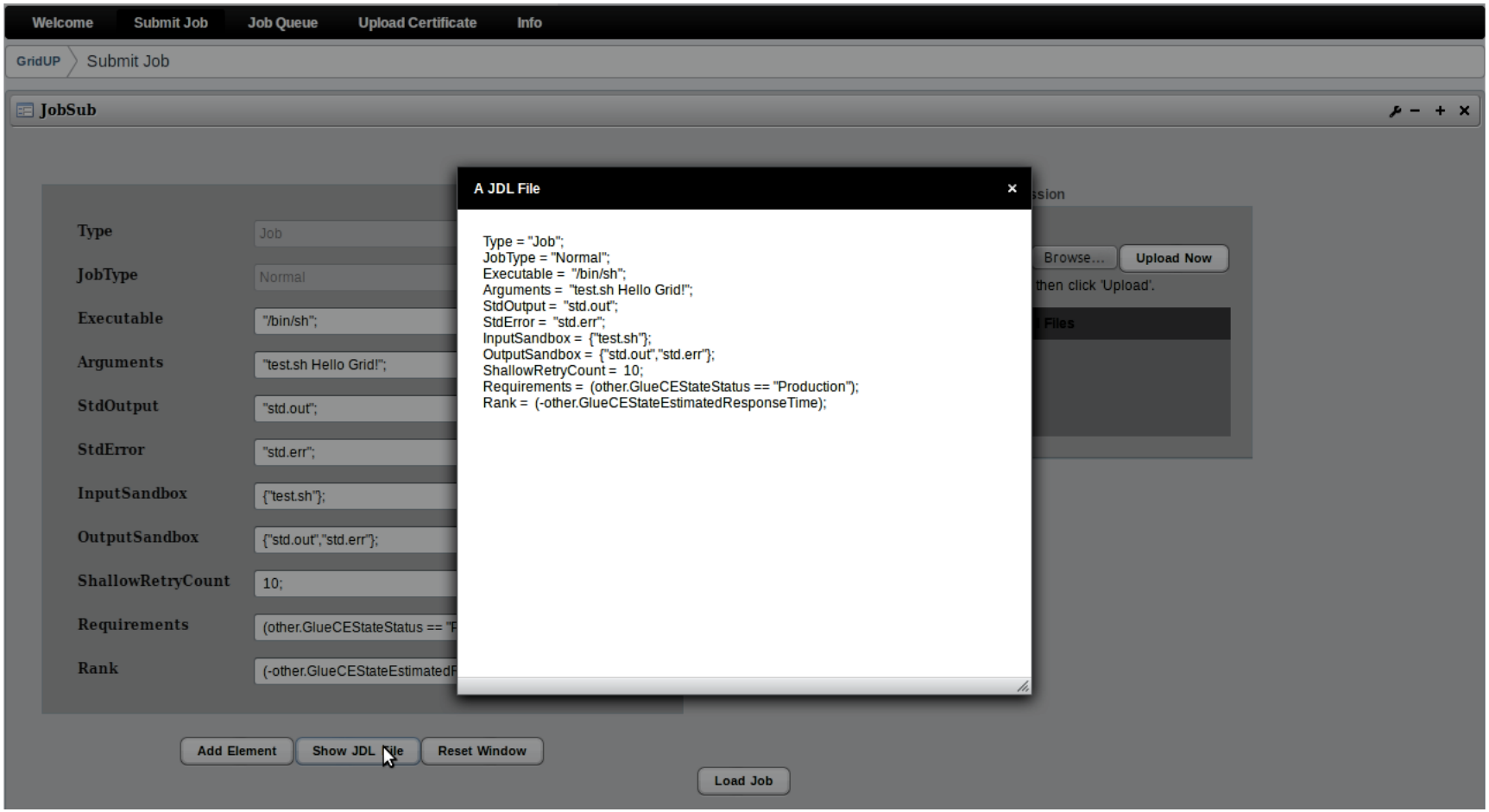

Figura A.13: Mostrar o ficheiro JDL

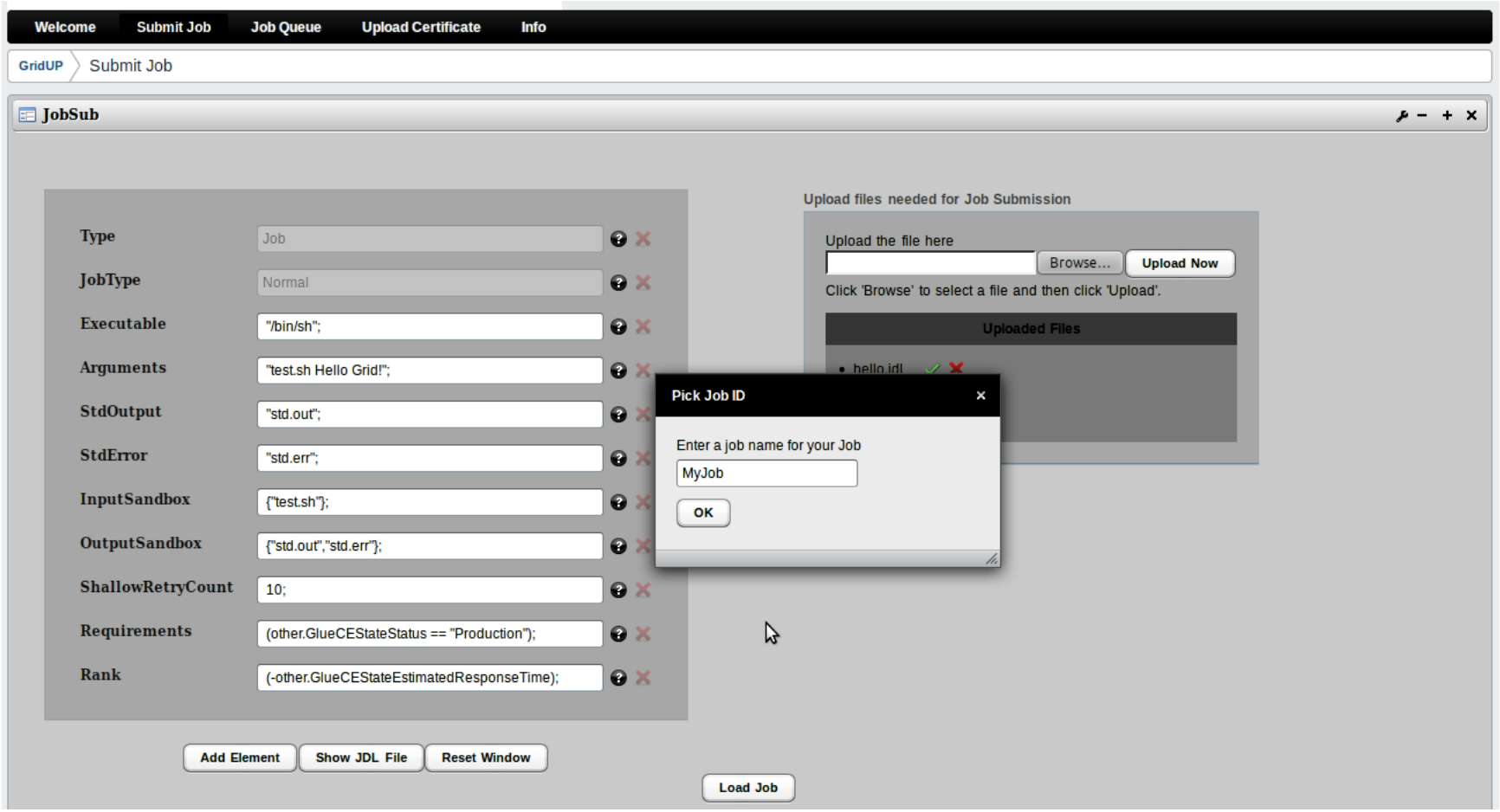

Figura A.14: Definir um nome para a tarefa

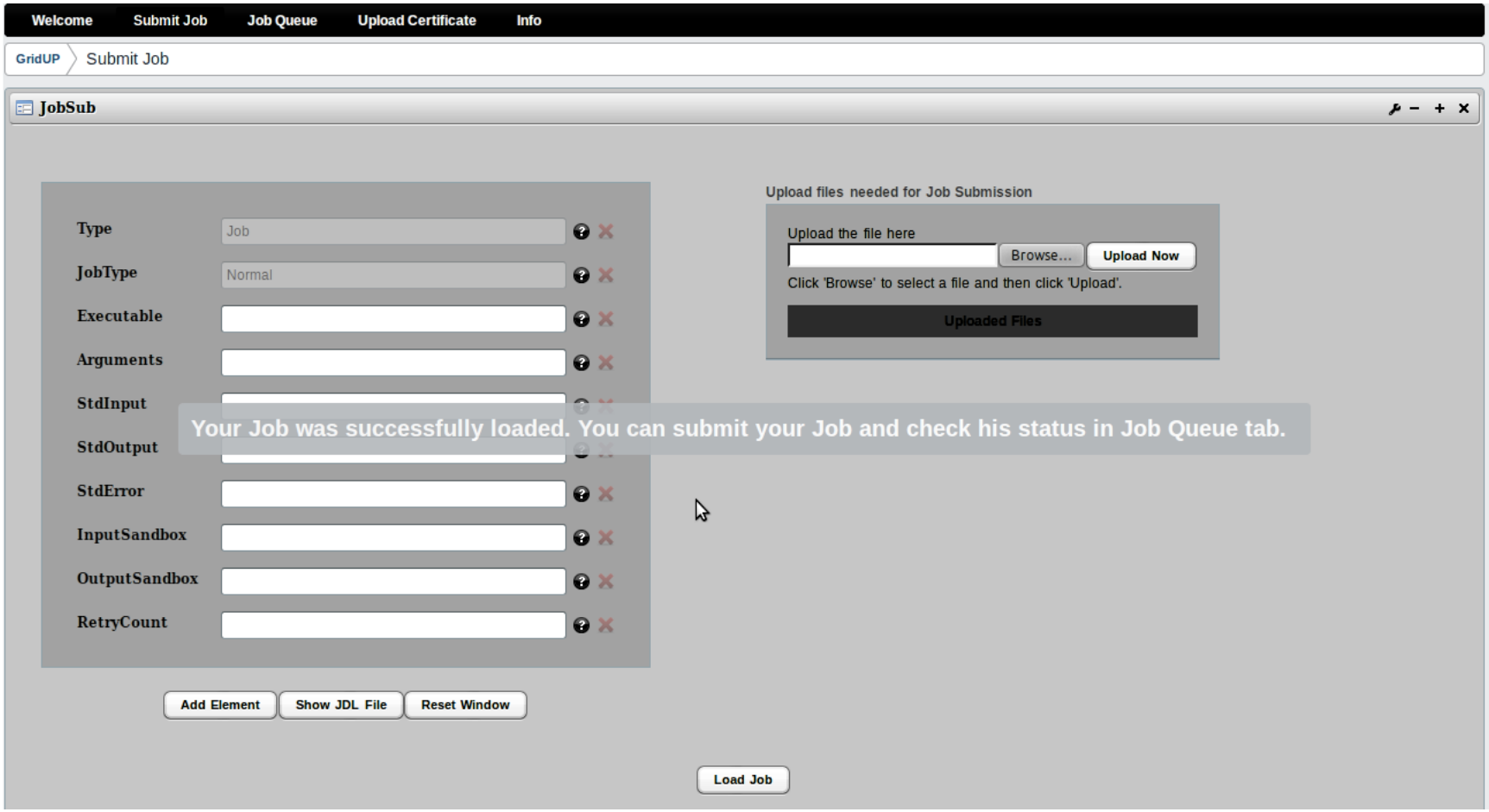

Figura A.15: Confirmação da submissão de uma tarefa

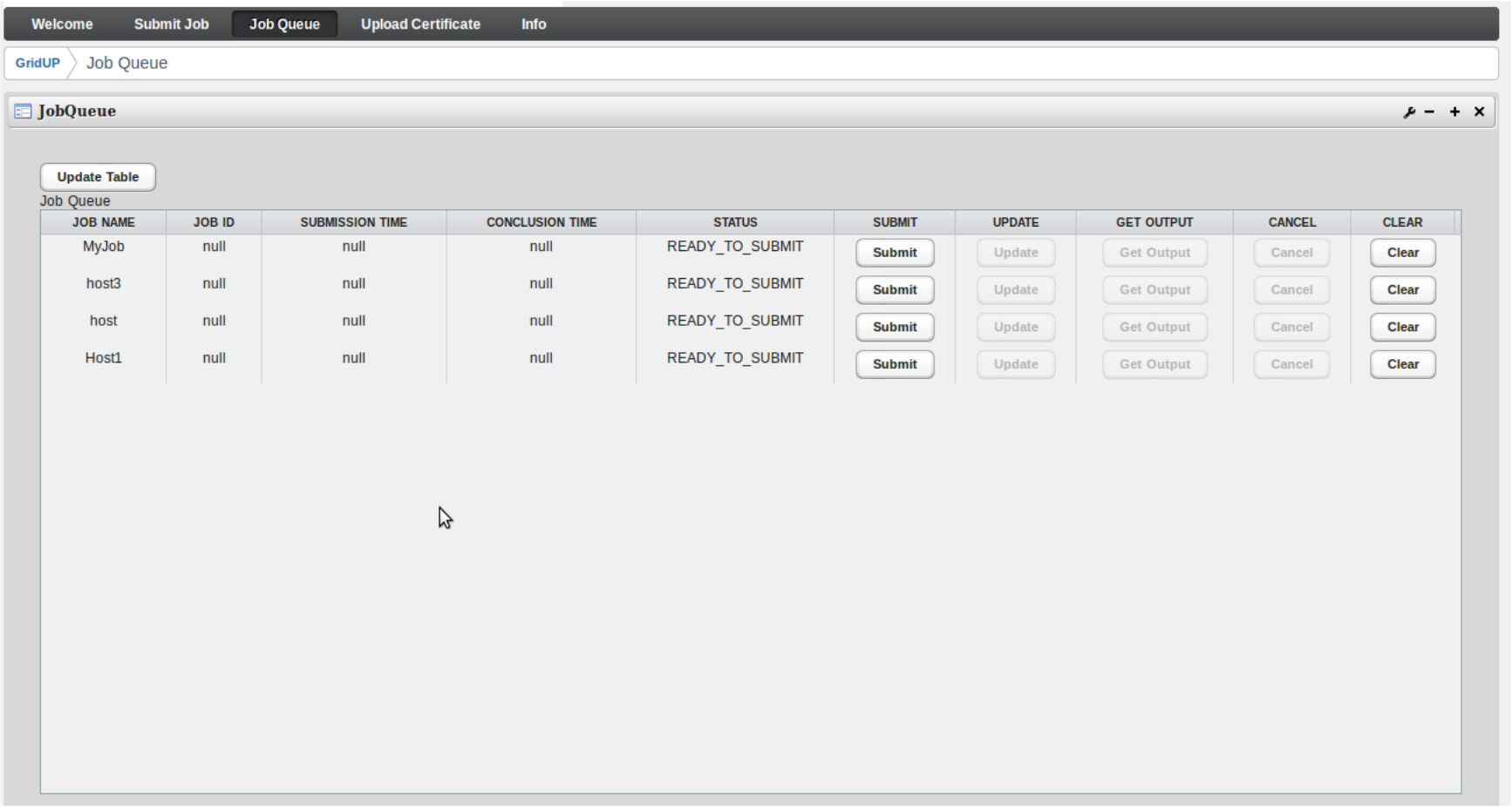

Figura A.16: Fila de tarefas

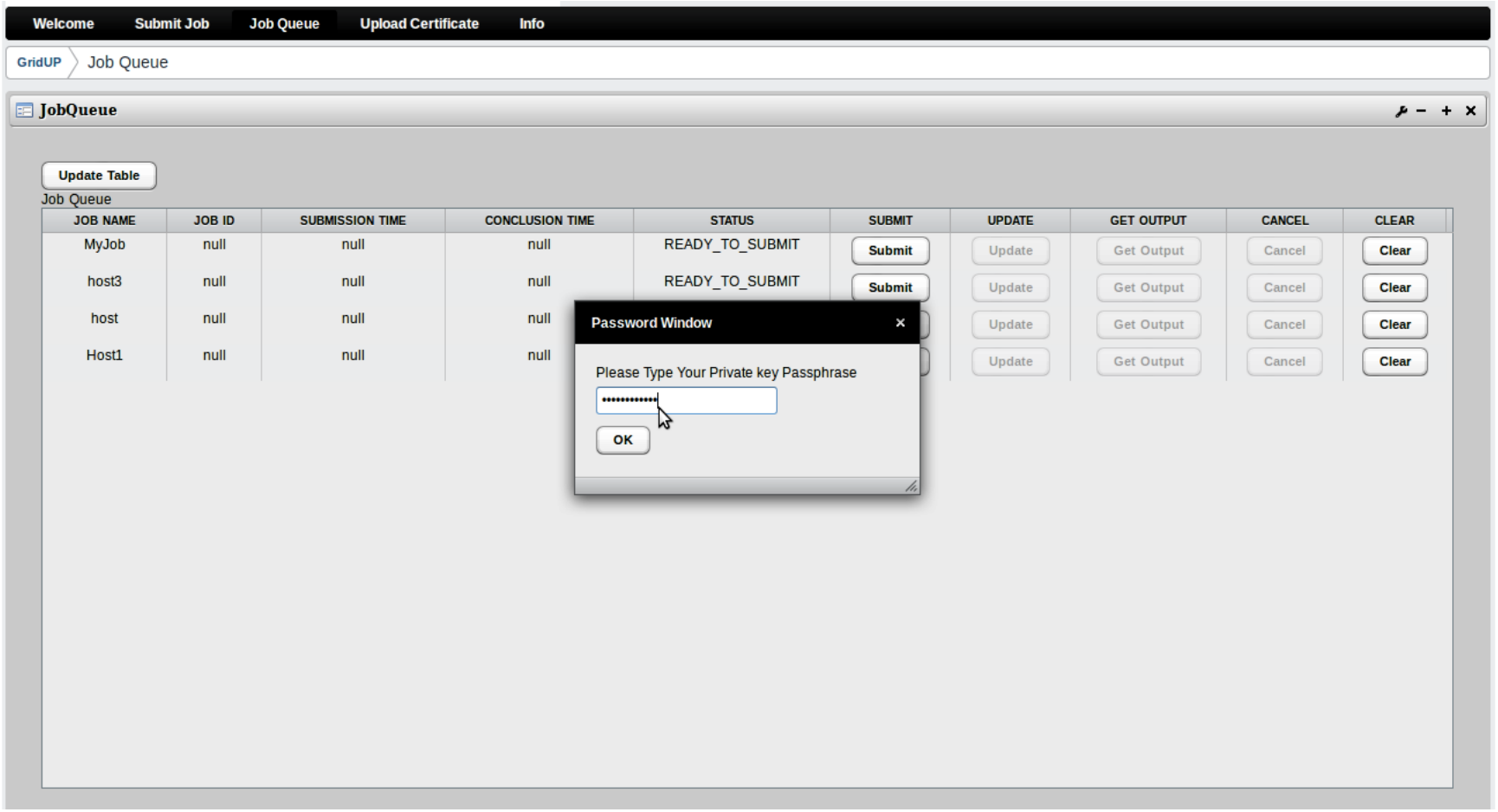

Figura A.17: Palavra chave - Certificado *proxy*

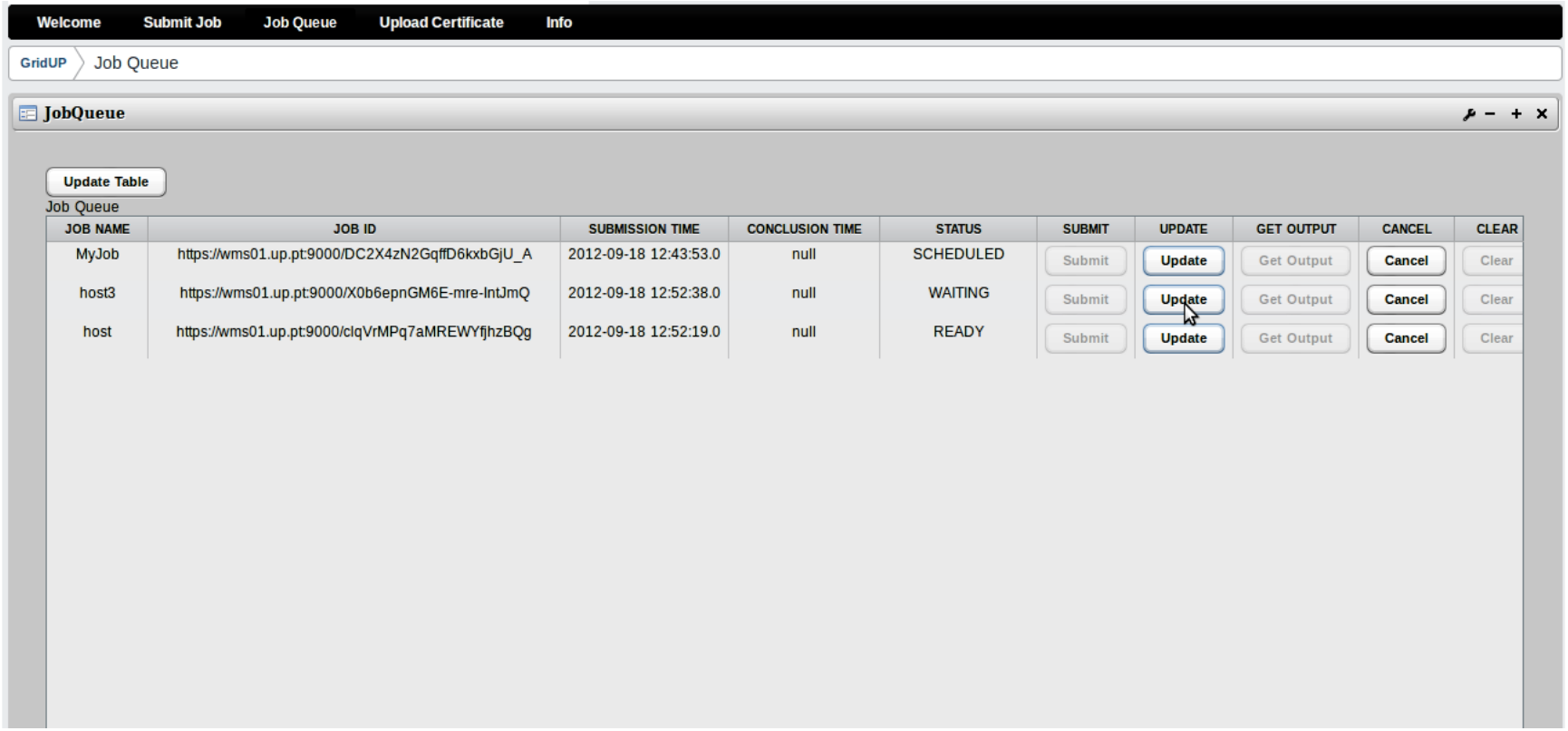

Figura A.18: Estado da fila de tarefas após submissão

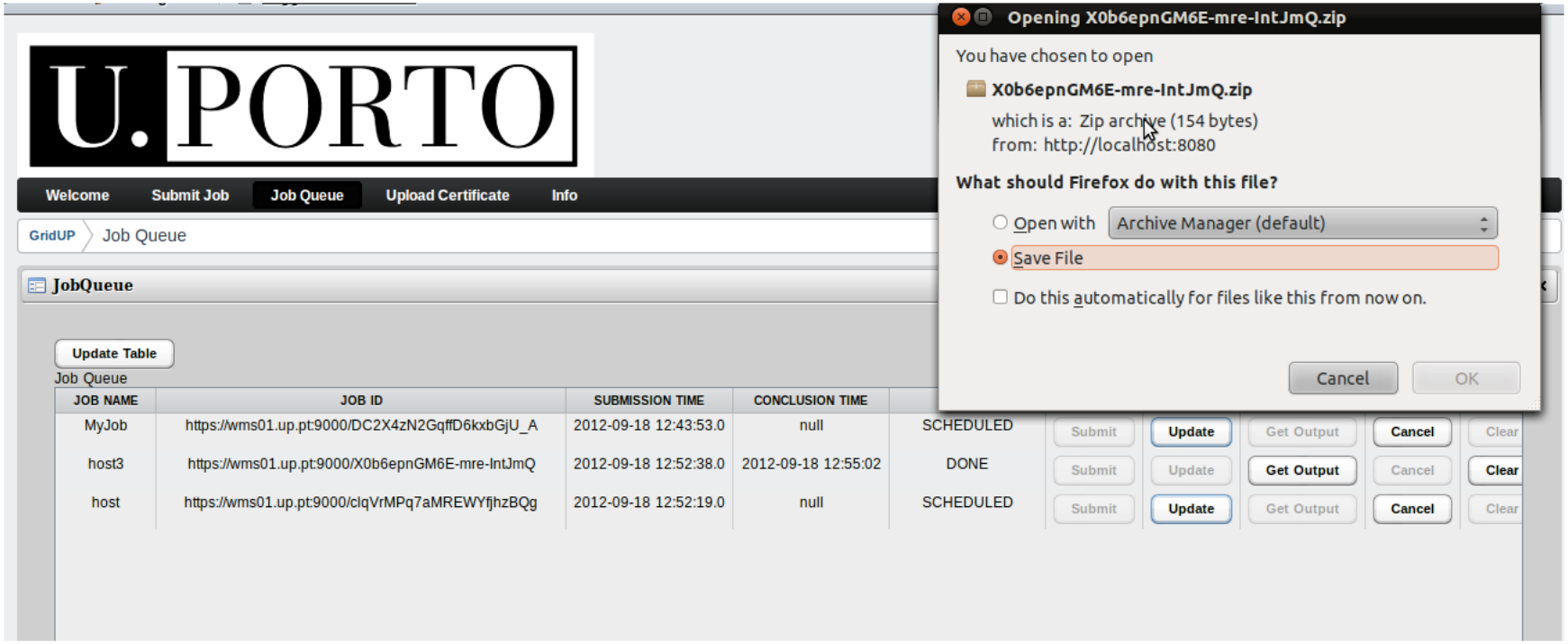

Figura A.19: Obter os resultados de uma tarefa

Referências

[1] Interactive european grid. ec.europa.eu/information_society/events/ict_bio_2006/docs/concert-meet-projects/inteugrid-w.pdf.

[2] glite r-gma information system, March 2007. http://glite.web.cern.ch/glite/packages/R3.0/deployment/glite-rgma-server-config/glite-rgma-server-config.asp.

[3] About the globus toolkit, September 2012. http://www.globus.org/toolkit/about.html.

[4] Apache tomcat, August 2012. http://tomcat.apache.org/tomcat-7.0-doc/index.html.

[5] Apel, February 2012. https://wiki.egi.eu/wiki/APEL#Description.

[6] Cgi - common gateway interface, July 2012. http://paginas.fe.up.pt/~goii2000/M9/cgi.htm.

[7] Cgi programming with perl, September 2012. http://docstore.mik.ua/orelly/linux/cgi/ch07_01.htm.

[8] Css introduction, August 2012. http://www.w3schools.com/css/css_intro.asp.

[9] Description of dag jobs, September 2012. http://wiki.egee-see.org/index.php/Description_of_DAG_jobs.

[10] Eclipse, September 2012. http://www.eclipse.org/org/.

[11] Egi-partners, September 2012. http://web.eu-egi.eu/partners/egi-council/http://web.eu-egi.eu/partners/egi-council/.

[12] Enginframe, September 2012. http://www.nice-software.com/products/enginframe.

[13] Eugridpma - building trust for authentication in e-science, September 2012. http://www.eugridpma.org/.

[14] Excerpt from beginning ajax, September 2012. http://www.wrox.com/WileyCDA/Section/id-303217.html.

[15] Expect - man page, August 2012. http://linuxcommand.org/man_pages/expect1.html.

[16] The free software foundation, August 2012. http://www.fsf.org/about/.

[17] Glassfish - community, August 2012. http://glassfish.java.net/docs/project.html.

[18] glibrary, September 2012. https://glibrary.ct.infn.it/glibrary/about2.php.

[19] Glite, September 2012. http://glite.cern.ch/introduction.

[20] glite voms server, March 2012. http://glite.web.cern.ch/glite/packages/R3.1/deployment/glite-VOMS_mysql/glite-VOMS_mysql.asp.

[21] Gnu operating system, September 2012. http://www.gnu.org/.

[22] Google web toolkit overview, August 2012. https://developers.google.com/web-toolkit/overview.

[23] Html - css, August 2012. http://www.w3.org/standards/webdesign/htmlcss#whatcss.

[24] Html introduction, July 2012. http://www.w3schools.com/html/html_intro.asp.

[25] The international grid trust federation, September 2012. http://www.igtf.net/.

[26] Introduction to vaadin development, August 2012. https://vaadin.com/learn.

[27] Is linux really more secure than windows?, September 2012. http://www.esecurityplanet.com/trends/article.php/3933491/Is-Linux-Really-More-Secure-than-Windows.htm.

[28] Javadoc, September 2012. http://agile.csc.ncsu.edu/SEMaterials/tutorials/javadoc/.

[29] Javaee, September 2012. http://www.oracle.com/technetwork/java/javaee/overview/index.html.

[30] Javascript introduction, July 2012. http://www.w3schools.com/js/js_intro.asp.

[31] Javaserver pages technology, July 2012. http://www.oracle.com/technetwork/java/javaee/jsp/index.html.

[32] Job collection - general description, September 2012. http://wiki.egee-see.org/index.php/Job_Collection.

[33] Liferay adopting the lgpl license, August 2012. http://www.liferay.com/web/bryan.cheung/blog/-/blogs/liferay-adopting-the-lgpl-license.

[34] The logging and bookeeping subsystem (lb), September 2012. http://glite.web.cern.ch/glite/lb/.

[35] Mpi on the grid: the batching environment - typical scripts, September 2012. http://wiki.egee-see.org/index.php/MPI_on_the_GRID:_the_batching_environment_-_typical_scripts.

[36] Mysql, August 2012. http://www.mysql.com/why-mysql/marketshare/.

[37] Mysql licensing policy, September 2012. http://www.mysql.com/about/legal/licensing/index.html.

[38] Nordugrid, September 2012. http://www.nordugrid.org/about.html.

[39] Ogf - overview, September 2012. http://www.gridforum.org/About/abt_overview.php.

[40] Ogf-arch, September 2012. http://www.cdac.in/html/events/beta-test/archives/gripsi-2007/gripsi-2007-gridcomp-overview.html.

[41] Oracle, August 2012. http://www.oracle.com/index.html.

[42] Parametric job - general description, September 2012. http://wiki.egee-see.org/index.php/Parametric_Jobs.

[43] Performance and scalability, September 2012. http://www.oracle.com/technetwork/database/performance/index.html.

[44] Perl intro, July 2012. http://perldoc.perl.org/perlintro.html.

[45] Php, portlets, August 2012. http://www.liferay.com/community/wiki/-/wiki/Main/PHP+Portlets.

[46] Pkcs 12: Personal information exchange syntax standard, September 2012. http://www.rsa.com/rsalabs/node.asp?id=2138.

[47] Project summary, August 2012. http://dev.mysql.com/doc/refman/5.6/en/introduction.html.

[48] Rich internet application, August 2012. http://www.statowl.com/custom_ria_market_penetration.php.

[49] Server reliability survey, September 2012. http://www.iaps.com/2008-server-reliability-survey.html.

[50] Shibboleth, September 2012. http://shibboleth.net/about/basic.html.

[51] Single grid architecture, July 2012. http://www.ucgrid.org/wiki/index.php/Single_Grid_Architecture.

[52] Tcl developer xchange, September 2012. http://www.tcl.tk/.

[53] The ucla grid portal, July 2012. www.csc.ucla.edu/Grid200408.pdf.

[54] Ugp readme, July 2012. http://www.ucgrid.org/wiki/index.php/README.

[55] Usage of operating systems for websites, September 2012. http://w3techs.com/technologies/overview/operating_system/all.

[56] Using a cluster from the ucla grid portal, July 2012. http://www.ats.ucla.edu/clusters/grid_portal/default.htm.

[57] Vaadin features, August 2012. https://vaadin.com/features.

[58] Virtual organization - diagram, September 2012. http://en.wikipedia.org/wiki/File:VirtOrg.png.

[59] What is a portlet?, July 2012. http://oreilly.com/pub/a/java/archive/what-is-a-portlet.html.

[60] What is a servlet?, September 2012. http://docs.oracle.com/javaee/6/tutorial/doc/bnafe.html.

[61] What is mpi?, September 2012. http://www.mcs.anl.gov/research/projects/mpi/.

[62] What is the p-grade grid portal, July 2012. http://portal.p-grade.hu/?m=home&s=0.

[63] What's ldap?, September 2012. http://tldp.org/HOWTO/LDAP-HOWTO/whatisldap.html.

[64] The workload management subsystem, September 2012. http://glite.web.cern.ch/glite/wms/.

[65] Yahoo! desktop widgets, August 2012. http://widgets.yahoo.com/.

[66] ¿qué es el science gateway?, July 2012. http://gisela-gw.ct.infn.it/sobre-el-sg.

[67] R. Barbera, A. Falzone, and A. Rodolico. The genius grid portal. Dipartimento di Fisica e Astronomia, INFN, and ALICE Collaboration Catania, Italy, March 2003. www.slac.stanford.edu/econf/C0303241/proc/papers/TUCT001.PDF.

[68] Roberto Barbera. Genius and gilda. Technical report, EGEE, May 2004. egee.cesnet.cz/en/user/genius.pdf.

[69] Roberto Barbera, Marco Fargetta, and Riccardo Rotondo. A simplified access to grid resources by science gateways. Department of Physics and Astronomy of the University of Catania and INFN, March 2011.

[70] Stephen Burke, Simone Campana, Elisa Lanciotti, Maarten Litmaath, Patricia Mendez Lorenzo, Vincenzo Miccio, Christopher Nater, Roberto Santinelli, and Andr a Sciaba. Glite 3.2 user guide. Technical report, CERN, 2011. Document identifier: CERN-LCG-GDEIS-722398.

[71] Tim Conrad. Postgresql vs mysql vs commercial databases: It's all about what you need. Technical report, Devx, 2004. `cosmoglobecorp.com/pdf/PostgreSQL_vs_MySQL_vs_DB2_vs_MSSQL_vs_Oracle.pdf`.

[72] Roberto Barbera et al. The genius grid portal and robot certificates: a new tool for e-science. volume 10. BMC Bioinformatics, June 2009. doi:10.1186/1471-2105-10-S6-S21.

[73] Z. Farkas and P. Kacsuk. P-grade portal: a generic workflow system to support user communities. Technical report, MTA SZTAKI, 1518 Budapest P.O. Box 63., Hungary, 2009. `portal.p-grade.hu/download/pgportal.pdf`.

[74] Ian Foster. What is the grid? a three point checklist. Technical report, Argonne National Laboratory & University of Chicago, July 2002. `http://dlib.cs.odu.edu/WhatIsTheGrid.pdf`.

[75] Ian Foster. Globus toolkit version 4: software for service-oriented systems. In *Proceedings of the 2005 IFIP international conference on Network and Parallel Computing*, NPC'05, pages 2–13, Berlin, Heidelberg, 2005. Springer-Verlag.

[76] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, August 2001.

[77] EGEE Group. Genius portal - overview. Technical report, EGEE. `http://egee.cesnet.cz/cms/export/sites/egee/en/user/genius-guide.pdf`.

[78] EGEE Group. Gilda grid demonstrator - overview. Technical report, EGEE. `http://egee.cesnet.cz/cms/export/sites/egee/en/user/Gilda_Overview.pdf`.

[79] UC Research Computing Group. The ucla grid architecture. Technical report, University of California. `http://www.ucgrid.org/wiki/index.php/Single_Grid_Architecture`.

[80] Marko Grönroos. *Book of Vaadin*. Vaadin Ltd, 4th edition, January 2012.

[81] Joshy Joseph and Craig Fellenstein. *Grid Computing*. On Demand Series. Pearson Education, Inc., 2004. ISBN:0-13-145660-1.

[82] Péter Kacsuk and Gergely Sipos. Multi-grid, multi-user workflows in the p-grade grid portal. *Journal of Grid Computing*, December 2005. DOI:10.1007/s10723-005-9012-6.

[83] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. *A taxonomy and survey of grid resource management systems for distributed computing*, volume 32. John Wiley & Sons, Ltd, February 2001. `www.buyya.com/papers/gridtaxonomy.pdf`.

[84] Maozhen Li and Mark Baker. *The Grid Core Technologies*. Wiley, 2005. ISBN-13 978-0-470-09417-4.

[85] Frédéric Magoulés, Jie Pan, Kiat-An Tan, and Abhinit Kumar. *Introduction to Grid Computing*. CRC Press, Taylor & Francis Group, 2009. ISBN:978-1-4200-7406-2.

[86] Paolo Missier, Stian Soiland-Reyes, Stuart Owen, Wei Tan, Aleksandra Nenadic, Ian Dunlop, Alan Williams, Thomas Oinn, and Carole Goble. Taverna, reloaded. In M. Gertz, T. Hey, and B. Ludaescher, editors, *SSDBM 2010*, Heidelberg, Germany, June 2010. `http://www.taverna.org.uk/pages/wp-content/uploads/2010/04/T2Architecture.pdf`.

[87] Zsolt Németh and Vaidy S. Sunderam. A comparison of conventional distributed computing environments and computational grids. In *International Conference on Computational Science (2)'02*, pages 729–738.

[88] Per Oster and Steven Newhouse. European grid infrastructure (egi). Technical report, European Union, 2011. `http://www.egi.eu/export/sites/egi/news-and-media/EGI_AnnualReport2011.pdf`.

[89] Jr. Richard Sezov. *Liferay in Action*. Manning Publications Co., 2012. ISBN 13: 978-1-935182-82-5 - ISBN 10: 1-935182-82-X.

[90] Guiseppe La Rocca. The glite api. `www.euasiagrid.org/dmdocuments/LAROCCA_API_PART_II.pdf`, May 2010.

[91] Richard L. Sezov, Jr., and Stephen Kostas. Portal administrator's guide. Technical report, Liferay Corp., 2010. `http://cdn.docs.liferay.com/portal/6.0/official/liferay-administrator-guide-6.0.pdf`.

[92] Joseph Shum and Alexander Chow et al. Liferay in action. Technical report, Liferay Corp., 2011. `http://docs.liferay.com/portal/6.0/official/liferay-developer-guide-6.0.pdf`.

[93] Larry Smarr and Charles E. Catlett. Metacomputing. *Commun. ACM*, 35(6):44–52, June 1992.

[94] Oleg Sukhoroslov. jlite user manual. Technical report, Institute for Systems Analysis, Russian Academy of Sciences, April 2010. `http://code.google.com/p/jlite/downloads/detail?name=jlite-manual-0.2.pdf`.

[95] Ian Taylor, Matthew Shields, Ian Wang, and Andrew Harrison. The Triana Workflow Environment: Architecture and Applications. In Ian Taylor, Ewa Deelman, Dennis Gannon, and Matthew Shields, editors, *Workflows for e-Science*, pages 320–339. Springer, New York, Secaucus, NJ, USA, 2007.

[96] Albert van Niekerk. Strategic management of media assets for optimizing market communication strategies, obtaining a sustainable competitive advantage and maximizing return on investment: An empirical study. 2006. `http://www.palgrave-journals.com/dam/journal/v3/n2/pdf/3650070a.pdf`.

[97] Barry Wilkinson. *Grid Computing, Techniques and Applications*. Computational Science Series. CRC Press, Taylor & Francis Group, 2010. ISBN:978-1-4200-6953-2.